Marina Blanton and Fattaneh Bayatbabolghani

Efficient Server-Aided Secure Two-Party Function Evaluation with Applications to Genomic Computation

Abstract: Computation based on genomic data is becoming increasingly popular today, be it for medical or other purposes. Non-medical uses of genomic data in a computation often take place in a server-mediated setting where the server offers the ability for joint genomic testing between the users. Undeniably, genomic data is highly sensitive, which in contrast to other biometry types, discloses a plethora of information not only about the data owner, but also about his or her relatives. Thus, there is an urgent need to protect genomic data. This is particularly true when the data is used in computation for what we call recreational non-health-related purposes. Towards this goal, in this work we put forward a framework for server-aided secure two-party computation with the security model motivated by genomic applications. One particular security setting that we treat in this work provides stronger security guarantees with respect to malicious users than the traditional malicious model. In particular, we incorporate certified inputs into secure computation based on garbled circuit evaluation to guarantee that a malicious user is unable to modify her inputs in order to learn unauthorized information about the other user's data. Our solutions are general in the sense that they can be used to securely evaluate arbitrary functions and offer attractive performance compared to the state of the art. We apply the general constructions to three specific types of genomic tests: paternity, genetic compatibility, and ancestry testing and implement the constructions. The results show that all such private tests can be executed within a matter of seconds or less despite the large size of one's genomic data.

Keywords: Genomic computation, garbled circuits, server-aided computation, certified inputs

DOI 10.1515/popets-2016-0033
Received 2016-02-29; revised 2016-06-02; accepted 2016-06-02.

1 Introduction

The motivation for this work comes from rapidly expanding availability and use of genomic data in a variety of applications and the need to protect such highly sensitive data from potential abuse. The cost of sequencing one's genome has dramatically decreased in the recent years and is continuing to decrease, which makes such data more readily available for a number of applications such as:

- *personalized medicine* uses genomic tests prior to prescribing a treatment to ensure its success;
- *paternity testing* uses DNA data to determine whether one individual is the father of another individual;
- *genomic compatibility tests* allow potential or current partners to determine whether their future children are likely to inherit genetic conditions;
- *determining ancestry and building genealogical trees* is done by examining DNA data of many individuals and finding relationships among specific individuals.

Genomic tests are increasingly used for medical purposes to ensure the best treatment. A number of services for what we call the "leisure" use of DNA data has flourished as well (e.g., [1–3]) allowing for various forms of comparing DNA data, be it for building ancestry trees, genomic compatibility or other.

It is clear that DNA is highly sensitive and needs to be protected from abuse. It can be viewed as being even more sensitive than other types of an individual's biometry, as not only does it allow for unique identification of the individual, but it also allows for learning a plethora of information about the individual such as predisposition to medical conditions and relatives of the individual thus exposing information about others as well. Furthermore, our understanding of genomes is continuously growing and exposure of DNA data now can lead to consequences which we cannot even anticipate today. For that reason, a number of publications that enable genomic computation while preserving privacy of DNA data have appeared in the literature (see, e.g., [9–12, 15, 29]). Such publications span several types of genomic computation including medical (such as personalized medicine and disease risk computation) and non-medical applications (such as paternity testing).

While protecting privacy of genomic data is important for all applications, in our opinion, it is more difficult for an individual to influence the way medical procedures are conducted than services in which individuals decide to voluntarily participate. That is, if genetic tests are necessary for a patient to determine the most effective treatment and the procedures in place are considered law-compliant, the patient has little possibility for influencing the way the DNA tests are conducted (besides, perhaps, declining to take the test and risking that the prescribed generic treatment is ineffective for her or has severe side effects). With what we consider as "leisure" uses of

Marina Blanton: University of Notre Dame, E-mail: mblanton@nd.edu
Fattaneh Bayatbabolghani: University of Notre Dame, E-mail: fbayatba@nd.edu

DNA information, the situation is different. An individual who meets a potential partner through a gene-based matchmaking online service (such as [3]) might be reluctant to share her DNA data with the service (or the partner) for the purpose of genetic compatibility tests. However, if the user is assured that no sensitive information about her DNA will be revealed to any party throughout the computation other than the intended outcome, she might revisit the decision to participate in such services. Thus, in the rest of this work, when we refer to genomic computation, we focus on applications which are not detrimental to the well-being of an individual and rather consider tests in which individuals might choose to participate.

The first observation we make about such types of genomic computation is that they are normally facilitated by some service or a third party. For example, both ancestry and gene-based matchmaking web sites allow participants to interact with each other through the service provider. Such service providers serve as a natural point for aiding the individuals with private computation on their sensitive genomic data. In some prior publications on genomic computation (e.g., [12]), it is assumed that computation such as paternity testing or genetic compatibility is run between a client and a server, while we believe that it is more natural to assume that such computation is carried out by two individuals through some third-party service provider. Thus, in this work we look at private genomic computation in the light of server-mediated setting and utilize the server to lower the cost of the computation for the participants. Throughout this work, we will refer to the participants as Alice (A), Bob (B), and the server (S).

From the security point of view, participants in a protocol that securely evaluates a function are normally assumed to be either semi-honest (also known as honest-but-curious or passive) or malicious (also known as active). In our application domain, we may want to distinguish between different security settings depending on how well Alice and Bob know each other. For example, if Alice and Bob are relatives and would like to know how closely they are related (i.e., how closely their genealogical trees overlap), it would be reasonable to assume that they will not deviate from the prescribed computation in the attempt to cheat each other, i.e., they can be assumed to be semi-honest. On the other hand, if Alice and Bob meet each other through a matchmaking web site and do not know each other well, it is reasonable for them to be cautious and engage in a protocol that ensures security (i.e., correctness and privacy) even in the presence of malicious participants. The server can typically be expected not to deviate from its prescribed behavior, as it would lose its reputation and consequently revenue if any attempts at cheating become known. If, however, adding protection against server's malicious actions is not very costly, it can also be meaningful to assume a stronger security model.

Another important consideration from a security point of view is enforcing correct inputs to be entered in the computation when, for instance, the inputs are certified by some authority. This requirement is outside the traditional security model for secure multi-party computation (even in the presence of fully malicious actors), and to the best of our knowledge certified inputs were previously considered only for specific functionalities such as private set intersection [22, 31] or anonymous credentials and certification [19], but not for general secure function evaluation (SFE). We bring this up in the context of genomic computation because for certain types of genomic tests it is very easy for one participant to modify his inputs and learn sensitive information about genetic conditions of the other party. For example, genetic compatibility tests evaluate the possibility of two potential or existing partners of transmitting to their children a genetic disease. Such possibility is present when both partners are (silent) carriers of that disease (see section 3.1 for more detail). Then if the partners can each separately evaluate their DNA for a fingerprint of a specific disease, the joint computation can consist of a simple AND of the bits provided by both parties (for one or more conditions). Now if a malicious participant sets all of his input bits to 1 and the outcome is positive, the participant learns that the other party is a carrier for a specific medical condition (or at least one condition from the set of specific conditions). We thus want to prevent malicious participants from modifying their inputs used in genomic computation in cases such data can be certified by certification authorities such as medical facilities.

In this work we address fairness, as one of the important properties of secure computation. In particular, it is known that full fairness cannot be achieved in the case of two-party computation in the malicious security model [26], but it becomes possible in the server-aided setting. Fairness has been considered in the server-aided literature in the past [38, 47] and achieving fairness only adds minimal overhead to the solutions in the settings we consider.

Contributions. While we draw motivation from genomic computation, our results are general and can be applied to any function. All constructions rely on garbled circuit evaluation typically used in the two-party setting (see section 3.2), but which we adopt to the three-party computation between the server and two users. Based on the motivation given above, we consider different adversarial settings, which we present from the simplest and enabling most efficient solutions to the most complex with added security.

1. Our most efficient solution is designed for the setting where A and B are semi-honest (and S can be malicious), as in ancestry testing. In this setting, the solution consists of a single circuit garbling and evaluation and the need for oblivious transfer (OT) is fully eliminated.

2. Our second solution assumes that A and B can be malicious, but S is semi-honest, as suitable for the paternity test, and achieves fairness for A and B. In this solution, the combined work for all participants is approximately the same as the combined work of two participants in a two-party protocol with semi-honest parties only.
3. Our last solution strengthens the model of malicious A and B with input certification (applicable to the genomic compatibility test). In more detail, in addition to being able to behave arbitrarily, A and B may maliciously modify their true inputs. To combat this, the function being evaluated is modified to mark any suitable subset of the inputs as requiring certification. At the time of secure function evaluation, A and B have to prove that the inputs they enter in the protocol are identical to the values signed by a trusted authority (a medical facility that performs genomic tests in our case). Achieving this involves the use of additional tools such as a signature scheme and zero-knowledge proofs of knowledge (ZKPKs). Handling of the remaining inputs and the rest of the computation is not affected by the shift to a stronger security model.

We assume that the participants do not collude.

All of our constructions offer conceptual simplicity and at the same time achieve highly attractive performance. The strongest of our models that enforces input correctness is novel and has not been treated in the context of general secure multi-party computation, and computation based on garbled circuits in particular. Despite the drastic differences in the techniques for garbled circuit evaluation and data certification, we show how they can be integrated by using OT as the connecting point or even when OT is not used.

Based on the solutions described above, we build implementations of three genetic tests, namely, genetic common ancestry, paternity, and genetic compatibility tests. Each test uses a different security setting. We show through experimental results that each test is efficient with the worst runtime being on the order of a couple of seconds. The performance favorably compares to the state of the art (as detailed in section 7), in some cases achieving orders of magnitude performance improvement over existing solutions.

2 Related Work

Literature on secure multi-party computation is extensive and cannot be covered here. In what follows, we concentrate on (i) secure server-aided two- or multi-party computation and (ii) work on privacy-preserving solutions for genetic tests.

Server-aided computation. The closest to our work is that of Herzberg and Shulman [38, 39] that considers two-party SFE based on garbled circuits with the aid of weakly trusted servers. The solution achieves security and fairness in the presence of malicious A and B. The authors also informally discuss (in [39]) extensions to guarantee security in the presence of malicious servers or collusion. Compared to that work, our solution in the presence of malicious A and B is more efficient in that that [38, 39] require the parties to perform $O(\kappa n)$ signature verifications and engage in $O(\kappa n)$ OTs, where κ is the security parameter and n is the number of (B's) inputs. The server's work is also larger than in our solution. The use of the server, however, is more constrained in [38, 39] (i.e., the server is used to answer queries of different types, but it does not participate in interactive computation).

Kamara et al. [47] assume a different setting, where a number of parties use a server to reduce computational burden for some of them. Using a solution based on garbled circuits, the work achieves work sublinear in the circuit size for some parties and work polynomial in the circuit size for the remaining parties and the server. Security holds when either the server and another party are malicious or when the server is semi-honest and all but one party are malicious. The model relies on non-colluding adversaries (termed non-cooperating adversaries in [46, 47]), who even when behaving maliciously do not collude with others. The work also addresses fairness. While not directly comparable to our result, the work of [47] uses what can be viewed as a more challenging security setting because all of our security settings assume a fixed semi-honest party and thus allow for more efficient constructions.

Beye et al. [14] supplement homomorphic encryption with server-aided garbled circuit evaluation for a number of building blocks using the solution from Kamara et al. [46] in the presence of semi-honest participants. The latter work provides a protocol that is similar to our first solution with semi-honest users (in performance and properties), but additionally involves coin tossing that requires public-key operations.

Carter et al. [24] use the aid of a server to reduce the cost of two-party SFE based on garbled circuits when any participant can be malicious. One party is assumed to be very weak (e.g., a mobile phone), while the second participant and the server are more powerful. The solution lifts most of the burden of two-party SFE in the presence of malicious participants from the weak party, but the work of the remaining parties is still comparable to the work in regular two-party SFE based on garbled circuits. Carter et al. [23] improve the result by building Whitewash, where the work performed by the weak party is further reduced. In addition, Mood et al. [60] present a solution in a similar outsourced setting where garbled outputs can be mapped to garbled inputs of another circuit to save on both computation and communication for some functions.

Kolesnikov et al. [50] consider the problem of input consistency in two-party SFE in the presence of malicious play-

ers with the aid of a semi-honest server. The goal is to ensure that both A and B enter the same input during multiple interactions, which is enforced with the help of the semi-honest server at low cost. This solution is not suitable for our goal of guaranteeing input correctness as a malicious participant can consistently provide incorrect inputs and by doing so violate privacy of possibly multiple users. Furthermore, there may not be multiple interactions between the same pair of users to enforce input consistency. That work also mentions the possibility of input certification in secure two-party computation, but we are not aware of realizations of this idea.

There are also publications [32, 43, 59] in the three-party setting that utilize garbled circuits. Feige et al. [32] studied minimal models for secure two-party computation and provided constructions for the setting in which a function f that produces a bit is to be evaluated on the inputs of parties A and B, but party C learns the result. Our first protocol for semi-honest A and B is similar to one of the constructions sketched in that work (see section 5.1). Two other concurrent to our work and independent publications [43, 59] study secure three-party computation in the presence of a single malicious party and offer efficient constructions based on garbled circuits. The solutions provided in both [43] and [59] are close in their efficiency to two-party protocols based on garbled circuits in the semi-honest setting, but neither can achieve fairness. In particular, [43] shows security in the selective abort model while [59] shows security in the standard model with abort.

Lastly, publications such as [25, 44] put forward generic constructions for outsourcing secure computation to multiple servers. Unlike this work, the focus is on enabling clients to verify the result of the computation with the overall cost being insignificantly higher than the cost of securely evaluating the function itself. [44] considers any number of clients and workers, while [25] treats two-party computation that substantially reduces the work of one party by employing an extra worker.

We summarize complexity of constructions from prior and our work in Table 1. In the table, u denotes the number of non-free gates in function f, κ_1 (κ_2) denotes a security parameter for symmetric (public key) cryptography, t_1 (t_2) is the number of A's (B's) input bits, t_3 is the number of output bits, t_1^c (t_2^c) is the number of certified bits in A's (B's) input, and $t_1^n = t_1 - t_1^c$ ($t_2^n = t_2 - t_2^c$) is the number of remaining input bits of A (B), σ and s are statistical security parameters (for the number of garbled circuits and encoding bits of an input bit in the malicious model). We use $\min(t, \kappa_1)$ public key operations for t (1-out-of-2) OTs with an OT extension and assume $\kappa_2 > \kappa_1$. The function is more complex in [38] and $u^* > u$. Similarly, $u' > u$ in [23].

Genomic computation. There are a number of publications, e.g., [9–11] and others, that treat the problem of privately computing personalized medicine tests with the goal of choosing an optimal medical treatment or drug prescription. Ayday et al. [8] also focus on privacy-preserving systems for storing genomic data by means of homomorphic encryption. Because personalized medicine is outside the scope of this work, we do not further elaborate on such solutions.

To the best of our knowledge, privacy-preserving paternity testing was first considered by Bruekers et al. in [15]. The authors propose privacy-preserving protocols for a number of genetic tests based on Short Tandem Repeats (STRs) (see section 3.1 for detail). The tests include identity testing, paternity tests with one and two parents, and common ancestry testing on the Y chromosome. The proposed protocols for these tests are based on additively homomorphic public key encryption and are secure in the presence of semi-honest participants. Implementation results were not given in [15], but Baldi et al. [12] estimates that the paternity test in [15] is several times slower than that in [12]. We thus compare our paternity test to the performance of an equivalent test in [12].

Baldi et al. [12] concentrate on a different representation of genomic data (in the form of fully-sequenced human genome) and provide solutions for paternity, drug testing for personalized medicine, and genetic compatibility. The solutions use private set intersection as the primary cryptographic building block in the two-party server-client setting. They were implemented and shown to result in attractive runtimes and we compare the performance of our paternity and compatibility tests to the results reported in [12] in section 7.

Related to that is the work of De Cristofaro et al. [28] that evaluates the possibility of using smartphones for performing private genetic tests. It treated paternity, ancestry, and personalized medicine tests. The protocol for the paternity test is the same as in [12] with certain optimizations for the smartphone platform (such as performing pre-processing on a more powerful machine). The ancestry test is performed by sampling genomic data as using inputs of large size deemed infeasible on a smartphone. The implementation also used private set intersection as the building block. Our implementation, however, can handle inputs of very large sizes at low cost.

Two recent articles [37, 40] describe mechanisms for private testing for genetic relatives and can detect up to fifth degree cousins. The solutions rely on fuzzy extractors. They encode genomic data in a special form and conduct testing on encoded data. The approach is not comparable to the solutions we put forward here as [37, 40] are based on non-interactive computation and is limited to a specific set of functions.

Although not as closely related to our work as publications that implement specific genetic tests, there are also publications that focus on applications of string matching to DNA testing. One example is the work of De Cristofaro et al. [30] that provides a secure and efficient protocol that hides the

Table 1. Complexity of constructions in prior and our work.

	Party	Communication	Sym. key/hash op.	Public key operations	Security model
[38], [39]	A	$O(\kappa_2(t_1 + s(t_2 + \kappa_1)) + t_3)$	$O(s(t_2 + \kappa_1))$	$O(\kappa_1)$	malicious A or B, fairness
	B	$O(\kappa_2(t_1 + s(\kappa_1 + t_2)) + \kappa_1(u^* + t_3))$	$O(u^* + s(t_2 + \kappa_1))$	$O(t_1 + s(t_2 + \kappa_1))$	
	S	$O(\kappa_2(t_1 + s(t_2 + \kappa_1)) + \kappa_1(u^* + t_3))$	$O(u^*)$	$O(t_1 + s(t_2 + \kappa_1))$	
[47]	A	$O(\kappa_1(\sigma \cdot t_1 + t_2 + t_3))$	$O(\sigma(t_1 + t_2))$	$-$	malicious S, semi-honest A & B OR semi-honest S, malicious A or B, fairness in [47] only
	B	$O(\kappa_1(\sigma(u + t_2) + t_1 + t_3))$	$O(\sigma(u + t_1 + t_2))$	$-$	
	S	$O(\kappa_1(\sigma \cdot u + t_1 + t_2 + t_3))$	$O(\sigma \cdot u)$	$-$	
[24], [60]	A	$O(\kappa_1(\sigma t_3 + st_1) + \kappa_2(\sigma t_2 + t_3 + k_1))$	$O(\sigma t_3)$	$O(\sigma t_3 + \kappa_1)$	
	B	$O(\kappa_1(\sigma(st_1 + t_2 + t_3)) + \kappa_2(\sigma t_2 + t_3 + k_1))$	$O(\sigma(u + st_1 + t_2 + t_3))$	$O(\sigma(t_2 + t_3) + \kappa_1)$	
	S	$O(\kappa_1(\sigma(st_1 + t_2 + t_3) + \kappa_2(\sigma t_2 + t_3))$	$O(\sigma(u + st_1 + t_3))$	$O(\sigma t_3)$	
[23]	A	$O(\kappa_1\sigma(t_1 + t_3 + \kappa_1))$	$O(\sigma(t_1 + t_3 + \kappa_1))$	$-$	malicious A & S or malicious B
	B	$O((\kappa_1(\sigma + \kappa_1)(t_1 + t_3 + \kappa_1) + \sigma u')$ $+t_2(\kappa_1 + t_2))$	$O((\sigma + \kappa_1)(t_3 + \kappa_1)$ $+\sigma u' + \kappa_1 t_1 + t_2)$	$O(\kappa_1)$	
	S	$O((\kappa_1(\sigma + \kappa_1)(t_1 + t_3 + \kappa_1) + \sigma u')$ $+t_2(\kappa_1 + t_2))$	$O((\sigma + \kappa_1)(t_1 + t_3 +$ $\kappa_1) + \sigma(u' + t_2))$	$O(\kappa_1)$	
[43]	any	$O(\kappa_1(u + t_3) +$ $\kappa_2(t_1 + t_2 + \min(t_1 + t_2, \kappa_1)))$	$O(u)$	$O(\min(t_1 + t_2, \kappa_1))$	one malicious party
[59]	any	$O(\kappa_1(u + t_1 + t_2 + t_3))$	$O(u + t_1 + t_2)$	$-$	
Protocol 1	A	$O(\kappa_1 \cdot t_1 + t_3)$	$-$	$-$	semi-honest A & B, mal. S, fairness
	B, S	$O(\kappa_1(u + t_1 + t_2 + t_3))$	$O(u)$	$-$	
Protocol 2	A	$O(\kappa_1(t_1 + t_3))$	$O(t_3)$	$-$	malicious A or B, semi-honest S, fairness
	B	$O(\kappa_1(u + t_2 + t_3) + \kappa_2 \cdot \min(t_2, \kappa_1))$	$O(u)$	$O(\min(t_2, \kappa_1))$	
	S	$O(\kappa_1(u + t_1 + t_2 + t_3) + \kappa_2 \cdot \min(t_2, \kappa_1))$	$O(u + t_3)$	$O(\min(t_2, \kappa_1))$	
Protocol 3	A	$O(\kappa_1(t_1 + t_3) + \kappa_2 \cdot t_1^c))$	$O(t_3)$	$O(t_1^c)$	as in protocol 2, plus certified inputs for A and B
	B	$O(\kappa_1(t_1 + t_2 + t_3) +$ $\kappa_2(t_2^c + \min(t_2^n, \kappa_1)))$	$O(u)$	$O(t_1^c + t_2^c +$ $\min(t_2^n, \kappa_1)$	
	S	$O(\kappa_1(u + t_1 + t_2 + t_3)$ $+\kappa_2(t_1^c + t_2^c + \min(t_2^n, \kappa_1)))$	$O(u + t_3)$	$O(t_1^c + t_2^c +$ $\min(t_2^n, \kappa_1))$	

size of the pattern to be searched and its position within the genome. Another example is the work of Katz et al. [48] that applies secure text processing techniques to DNA matching.

3 Preliminaries

3.1 Genomic testing

We next describe paternity, genetic compatibility, and ancestry tests. Genomic background can be found in Appendix A.

Paternity test. This test is normally based on STRs (see Appendix A). One's STR profile consists of an ordered sequence of N 2-element sets $S = \langle\{x_{1,1}, x_{1,2}\}, \{x_{2,1}, x_{2,2}\}, \ldots, \{x_{N,1}, x_{N,2}\}\rangle$, where each value corresponds to the number of repeats of a specific STR sequence at specific locations in the genome. For each STR i, one of $x_{i,1}$ and $x_{i,2}$ is inherited from the mother and one from the father.

Thus in the paternity test with a single parent, there are two STR profiles $S = \langle\{x_{i,1}, x_{i,2}\}\rangle$ and $S' = \langle\{x'_{i,1}, x'_{i,2}\}\rangle$ corresponding to the child and the contested father, respectively. To determine whether S' corresponds to the child's father, the test computes whether for each i the set $\{x_{i,1}, x_{i,2}\}$ contains (at least) one element from the set $\{x'_{i,1}, x'_{i,2}\}$. In other words, the test corresponds to the computation

$$\bigwedge_{i=1}^{N} [\{x_{1,i}, x_{2,i}\} \cap \{x'_{1,i}, x'_{2,i}\} \neq \emptyset] = \text{True} \qquad (1)$$

When testing with both parents is performed, for each STR i one of $x_{i,1}$ and $x_{i,2}$ must appear in the mother's set and the other in the father's set. Using both parents' profiles increases the accuracy of the test, but even the single parent test has high accuracy for a small number N of well-chosen STRs (e.g., the US CODIS system utilizes $N = 13$, while the European SGM Plus identification method uses $N = 10$).

Genetic compatibility test. Here we are interested in the genetic compatibility test where potential (or existing) partners would like to determine the possibility of transmitting to their children a genetic disease with Mendelian inheritance. In particular, if a specific mutation occurs in one allele, one of the alternative gene versions at a given location (called minor), it often has no impact on one's quality of life, but when the mutation occurs in both alleles (called major), the disease manifests itself in severe forms. If both partners silently carry a single mutation, they have a noticeable chance of conceiving a child carrying the major variety. Thus, a genetic compatibility

test for a given genetic disease would test for the presence of minor mutations in both partners.

The current practice for screening for most genetic diseases consists of testing one SNP in a specific gene. It is, however, expected that in the future tests for more complex diseases (that involve multiple genes and mutations) will become available. Thus, a genetic disease can be characterized by a set of SNP indices and the corresponding values $(i_1, b_1), \ldots, (i_t, b_t)$, where i_j is the SNP index and $b_j \in \{0, 1\}$ is the value it takes. Then if the same values are found in the appropriate SNPs of an individual, the individual is tested as positive (i.e., the individual is the disease carrier). If both partners test as positive, then the outcome of the genetic compatibility test will be treated as positive and otherwise it is negative.

Ancestry test. There are a number of tests that allow for various forms of ancestry testing, for example, tests using Y-chromosome STRs (applicable to males only), mitochondrial DNA (mtDNA) test on the maternal line, and more general SNP-based tests for common ancestry or one's genealogy. Many such tests are not standardized and current ancestry and genealogy service providers often use proprietary algorithms. The advantage of STR-based tests is that normally only a relatively small number of STRs are tested, while SNP-based tests often utilize a large number of (or even all available) SNPs, but more distant ancestry can be learned from SNP-based tests. For improved accuracy it is also possible to perform one type of testing after the other. In either case, to determine the most recent common ancestor between two individuals, the markers from the two individuals are compared and their number determines how closely the individuals are related. Certain tests such as determining geographical regions of one's ancestors normally require genetic data from many individuals.

3.2 Garbled circuit evaluation

The use of garbled circuits allows two parties P_1 and P_2 to securely evaluate a Boolean circuit of their choice. That is, given an arbitrary function $f(x_1, x_2)$ that depends on private inputs x_1 and x_2 of P_1 and P_2, respectively, the parties first represent it as a Boolean circuit. One party, say P_1, acts as a circuit generator and creates a garbled representation of the circuit by associating both values of each binary wire i (including input and output wires) with random labels ℓ_i^0 and ℓ_i^1. The other party, P_2, acts as a circuit evaluator and evaluates the circuit in its garbled representation without knowing the meaning of the labels that it handles during the evaluation. The output labels can be mapped to their meaning and revealed to either or both parties. Additional details can be found in Appendix A.

The basic approach is secure in the presence of a semi-honest circuit generator and a malicious evaluator [36] (and the knowledge of valid labels for the output wires implicitly proves that the computation was performed correctly [35]). However, extending the security to the malicious setting (when either party can be malicious) requires additional techniques which substantially degrade performance of the approach.

An important component of garbled circuit evaluation is 1-out-of-2 OT. It allows the circuit evaluator to obtain wire labels corresponding to its inputs. In particular, in OT the sender (i.e., circuit generator in our case) possesses two strings s_0 and s_1 and the receiver (circuit evaluator) has a bit σ. OT allows the receiver to obtain string s_σ and the sender learns nothing. An OT extension allows any number of OTs to be realized with small additional overhead per OT after a constant number of regular more costly OT protocols (the number of which depends on the security parameter). The literature contains many realizations of OT and its extensions, including recent work, but in this work we primarily are interested in OT protocols and OT extensions secure in the presence of malicious participants (such as [42, 61, 62] and others).

The fastest currently available approach for circuit generation and evaluation we are aware of is by Bellare et al. [13]. It is compatible with earlier optimizations, most notably the "free XOR" gate technique [52] that allows XOR gates to be processed without cryptographic operations or communication, resulting in virtually no overhead for such gates.

3.3 Signature schemes with protocols and commitment schemes

Our solution that enforces input correctness by means of user input certification relies on additional building blocks, which are signature schemes with protocols, commitment schemes, and zero-knowledge proofs of knowledge.

From the available signature schemes, e.g., [16, 17] with the ability to prove knowledge of a signature on a message without revealing the message, the Camenisch-Lysyanskaya scheme [16] is of interest to us. It uses public keys of the form (n, a, b, c), where n is an RSA modulus and a, b, c are random quadratic residues in $\mathbb{Z}_n^*$. A signature on message m is a tuple (e, s, v), where e is prime, e and s are randomly chosen according to security parameters, and v is computed to satisfy $v^e \equiv a^m b^s c \pmod{n}$. A signature can be issued on a block of messages. To sign a block of t messages $m_1, \ldots, m_t$, the public key needs to be of the form $(n, a_1, \ldots, a_t, b, c)$ and the signature is (e, s, v), where $v^e \equiv a_1^{m_1} \cdots a_t^{m_t} b^s c \pmod{n}$.

Given a public verification key (n, a, b, c), to prove knowledge of a signature (e, s, v) on a secret message m, one forms a commitment $c = \mathsf{Com}(m)$ and proves that she possesses a signature on the value committed in c (see [16] for detail). The commitment c can consecutively be used to prove

additional statements about m in zero knowledge. Similarly, if one wants to prove statements about multiple messages included in a signature, multiple commitments will be formed.

The commitment scheme used in [16] is that of Damgård and Fujisaki [27]. The setup consists of a public key (n, g, h), where n is an RSA modulus, h is a random quadratic residue in $\mathbb{Z}_n^*$, and g is an element in the group generated by h. The modulus n can be the same as or different from the modulus used in the signature scheme. For simplicity, we assume that the same modulus is used. To produce a commitment to x using the key (n, g, h), one randomly chooses $r \in \mathbb{Z}_n$ and sets $\mathsf{Com}(x, r) = g^x h^r \bmod n$. When the value of r is not essential, we may omit it and use $\mathsf{Com}(x)$ instead. This commitment scheme is statistically hiding and computationally binding. The values x, r are called the opening of $\mathsf{Com}(x, r)$.

Zero-knowledge proofs of knowledge (ZKPKs) allow one to prove a particular statement about private values without revealing additional information besides the statement itself. Following [20], we use notation $PK\{(vars) : statement\}$ to denote a ZKPK of the given statement, where the values appearing in the parentheses are private to the prover and the remaining values used in the statement are known to both the prover and verifier. If the proof is successful, the verifier is convinced of the statement of the proof. For example, $PK\{(\alpha) : y = g_1^\alpha \vee y = g_2^\alpha\}$ denotes that the prover knows the discrete logarithm of y to either the base g_1 or g_2. Lastly, because a proof of knowledge of a signature is cumbersome to write in this detailed form, we use abbreviation $\mathsf{Sig}(x)$ and $\mathsf{Com}(x)$ to indicate the knowledge of a signature and commitment, respectively. For example, $PK\{(\alpha) : \mathsf{Sig}(\alpha) \wedge y = \mathsf{Com}(\alpha) \wedge (\alpha = 0 \vee \alpha = 1)\}$ denotes a proof of knowledge of a signature on a bit committed to in y. Because proving the knowledge of a signature on x in [16] requires a commitment to x (which is either computed as part of the proof or may already be available from prior computation), we explicitly include the commitment into all proofs of a signature.

4 Security Model

We formulate security using the standard ideal/real model for secure multi-party computation, where the view of any adversary in the real protocol execution should be indistinguishable from its view in the ideal model where a trusted party (TP) evaluates the function. Because the server does not contribute any input, it is meaningful to consider that either A or B is honest since the goal is to protect the honest party.

As previously mentioned, we are primarily interested in the setting where the server is semi-honest, but parties A and B may either be semi-honest or fully malicious. Thus, we target security models where S complies with the computation, with the exception of the first setting of semi-honest A and B, where we get security in the presence of a malicious server for free. We similarly assume that the server will not collude with users (putting its reputation at risk) or let users affect its operation.

We obtain security settings where (1) A and B can be corrupted by a semi-honest adversary, while S can act on behalf of a fully malicious adversary and (2) A and B can be malicious, but the server is semi-honest. Because we assume that the parties (or the adversaries who corrupt them) do not collude, at any given point of time there might be multiple adversaries, but they are independent of each other. This is similar to the setting used in [46, 47]. We note that based on the security settings listed above, at most one adversary would be fully malicious. In other words, if in (2) A is malicious, the goal is to protect B who is assumed to not be malicious and S is semi-honest, while in (1) S can be malicious, while A and B are semi-honest. Kamara et al. [46], however, show that in the presence of non-cooperating adversaries who corrupt only one party, showing security can be reduced to showing that the protocol is secure in the presence of semi-honest adversaries only, followed by proving for each malicious adversary $\mathcal{A}_i$ that the solution is secure in the presence of $\mathcal{A}_i$ when all other parties are honest. More precisely, we rely on the following lemma:

Lemma 1 ([46]). *If a multi-party protocol Π between n parties $P_1, \ldots, P_n$ securely computes f in the presence of (i) independent and semi-honest adversaries and (ii) a malicious $\mathcal{A}_i$ and honest $\{\mathcal{A}_j\}_{j \neq i}$, then Π is also secure in the presence of an adversary $\mathcal{A}_i$ that is non-cooperative with respect to all other semi-honest adversaries.*

This implies that in our setting (2) a solution secure in the presence of malicious A or B will also remain secure when A and B are corrupted by two independent malicious adversaries.

To model fairness, we modify the behavior of the TP in the ideal model to send $\perp$ to all parties if any party chooses to abort (note that fairness is only applicable to A and B). We assume that A and B learn the result of evaluation of a predefined function f that takes input x_1 from A and x_2 from B, and the server learns nothing. Because our primary motivation is genomic computation, we consider single-output functions, i.e., both A and B learn $f(x_1, x_2)$ (but two of our constructions support functions where A's and B's outputs differ and the remaining protocol in the present form loses only fairness).

Execution in the real model. The execution of protocol Π in the real model takes place between parties A, B, S and a subset of adversaries $\mathcal{A}_A$, $\mathcal{A}_B$, $\mathcal{A}_S$ who can corrupt the corresponding party. Let $\mathcal{A}$ denote the set of adversaries present in a given protocol execution. A and B receive their respective inputs x_i and a set of random coins r_i, while S receives only a set of

random coins r_3. All parties also receive security parameter 1^κ. Each adversary receives all information that the party it corrupted has and a malicious adversary can also instruct the corresponding corrupted party to behave in a certain way. For each $\mathcal{A}_X \in \mathcal{A}$, let $\text{VIEW}_{\Pi,\mathcal{A}_X}$ denote the view of the adversary $\mathcal{A}_X$ at the end of an execution of Π. Also let $\text{OUT}^{\text{hon}}_{\Pi,\mathcal{A}}$ denote the output of the honest parties (if any) after the same execution of the protocol. Then for each $\mathcal{A}_X \in \mathcal{A}$, we define the partial output of a real-model execution of Π between A, B, S in the presence of $\mathcal{A}$ by $\text{REAL}_{\Pi,\mathcal{A}_X}(\kappa, x_1, x_2, r_1, r_2, r_3) \stackrel{\text{def}}{=} \text{VIEW}_{\Pi,\mathcal{A}_X} \cup \text{OUT}^{\text{hon}}_{\Pi,\mathcal{A}}$.

Execution in the ideal model. In the ideal model, all parties interact with a TP party who evaluates f. Similar to the real model, the execution begins with A and B receiving their respective inputs x_i and each party (A, B, and S) receiving security parameter 1^κ. Each honest (semi-honest) party sends to the TP $x'_i = x_i$ and each malicious party can send an arbitrary value x'_i to the TP. If x_1 or x_2 is equal to $\perp$ (empty) or if the TP receives an abort message, the TP returns $\perp$ to all participants. Otherwise, A and B receive $f(x'_1, x'_2)$. Let $\text{OUT}^{\text{hon}}_{f,\mathcal{A}}$ denote the output returned by the TP to the honest parties and let $\text{OUT}_{f,\mathcal{A}_X}$ denote the output that corrupted party $\mathcal{A}_X \in \mathcal{A}$ produces based on an arbitrary function of its view. For each $\mathcal{A}_X \in \mathcal{A}$, the partial output of an ideal-model execution of f between A, B, S in the presence of $\mathcal{A}$ is denoted by $\text{IDEAL}_{f,\mathcal{A}_X}(\kappa, x_1, x_2) \stackrel{\text{def}}{=} \text{OUT}_{f,\mathcal{A}_X} \cup \text{OUT}^{\text{hon}}_{f,\mathcal{A}}$.

Definition 1 (Security)**.** *A three-party protocol Π between A, B, and S securely computes f if for all sets of probabilistic polynomial time (PPT) adversaries $\mathcal{A}$ in the real model, for all x_i and $\kappa \in \mathbb{Z}$, there exists a PPT transformation $\mathcal{S}_X$ for each $\mathcal{A}_X \in \mathcal{A}$ such that $\text{REAL}_{\Pi,\mathcal{A}_X}(\kappa, x_1, x_2, r_1, r_2, r_3) \stackrel{c}{\approx} \text{IDEAL}_{f,\mathcal{S}_X}(\kappa, x_1, x_2)$, where each r_i is chosen uniformly at random and $\stackrel{c}{\approx}$ denotes computational indistinguishability.*

To model the setting where some of the inputs of A and/or B are certified, we augment the function f to be executed with the specification of what inputs are to be certified and two additional inputs y_1 and y_2 that provide certification for A's and B's inputs, respectively. Then in the ideal model execution, the TP will be charged with additionally receiving y_i's. If the TP does not receive all inputs or if upon receiving all inputs some inputs requiring certification do not verify, it sends $\perp$ to all parties. In the real model execution, verification of certified inputs is built into Π and besides using two additional inputs y_1 and y_2 the specification of the execution remains unchanged.

Definition 2 (Security with certified inputs)**.** *A three-party protocol Π between A, B, and S securely computes f if for all sets of PPT adversaries $\mathcal{A}$ in the real model, for all x_i, y_i, and $\kappa \in \mathbb{Z}$, there exists a PPT transformation $\mathcal{S}_X$ for each $\mathcal{A}_X \in \mathcal{A}$ such that $\text{REAL}_{\Pi,\mathcal{A}_X}(\kappa, x_1, x_2, y_1, y_2, r_1, r_2, r_3) \stackrel{c}{\approx} \text{IDEAL}_{f,\mathcal{S}_X}(\kappa, x_1, x_2, y_1, y_2)$, where each r_i is chosen uniformly at random.*

5 Server-Aided Computation

In this section we detail our solutions for server-aided two party computation based on garbled circuits. The current description is general and can be applied to any function f. In section 6 we describe how these constructions can be applied to genomic tests to result in fast performance.

5.1 Semi-honest A and B, malicious S

Our first security setting is where A and B are semi-honest and S can be malicious. The main intuition behind the solution is that when A and B can be assumed to be semi-honest and a solution based on garbled circuit evaluation is used, we will charge S with the task of evaluating a garbled circuit. That is, security is maintained in the presence of malicious server because garbled circuit evaluation techniques are secure in the presence of a malicious evaluator. Next, we notice that if A and B jointly form garbled representation of the circuit for the function f they would like to evaluate, both of them can have access to the pairs of labels (ℓ_i^0, ℓ_i^1) corresponding to the input wires. Thus, they can simply send the appropriate label ℓ_i^b to S for evaluation purposes for their value of the input bit b for each input wire. This eliminates the need for OT and results in a solution that outperforms a two-party protocol in the presence of only semi-honest participants. The same idea was sketched in [32] (with the difference that S was to learn the output). The use of a pseudo-random function $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$ with security parameter κ for deriving wire labels in the scheme is as in [59].

A more detailed description of the solution, which we denote as Protocol 1, is given next. In what follows, let m denote the total number of wires in a circuit (including input and output wires), wires $1, \ldots, t_1$ correspond to A's input, wires $t_1+1, \ldots, t_1+t_2$ correspond to B's input, and the last t_3 wires $m-t_3+1, \ldots, m$ correspond to the output wires. We also use κ to denote security parameter (for symmetric key cryptography). Notation $a \stackrel{R}{\leftarrow} U$ means that the value of a is chosen uniformly at random from the set U. The protocol is written to utilize the free XOR technique, where $\ell_i^0 \oplus \ell_i^1$ must take the same value Δ for all circuit wires i and the last bit of Δ is 1.

In Protocol 1, the easiest way for A and B to jointly choose random values is for one party to produces them and commu-

Input: A has private input x_1, B has private input x_2, and S has no private input.
Output: A and B learn $f(x_1, x_2)$, S learns nothing.
Protocol 1:

1. A and B jointly choose $\delta \xleftarrow{R} \{0,1\}^{\kappa-1}$, $k \xleftarrow{R} \{0,1\}^{\kappa}$, and set $\Delta = \delta||1$. They jointly produce m pairs of garbled labels as $\ell_i^0 = \mathsf{PRF}(k, i)$ and $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [1, m]$, garble the gates to produce garbled circuit $\mathcal{G}_f$ for f, and send $\mathcal{G}_f$ to S.
2. For each $i \in [1, t_1]$, A locates the ith bit b_i of her input and sends to S the label $\ell_i^{b_i}$ of the corresponding wire i in the garbled circuit.
3. Similarly, for each bit $j \in [1, t_2]$, B locates the jth bit b_j of his input and sends to S the label $\ell_{i+t_1}^{b_j}$ of the corresponding wire $i + t_1$ in the garbled circuit.
4. S evaluates the circuit on the received inputs and returns to B the computed label ℓ_i^b for each output wire $i \in [m - t_3 + 1, m]$. B forwards all received information to A.
5. For each ℓ_i^b returned by S ($i \in [m - t_3 + 1, m]$), A and B do the following: if $\ell_i^b = \ell_i^0$, set $(i - m + t_3)$th bit of the output to 0, if $\ell_i^b = \ell_i^1$, set $(i - m + t_3)$th bit of the output to 1, otherwise abort.

nicate to the other party. In this solution, the combined work of A and B is linear in the size of the circuit for f. The work, however, can be distributed in an arbitrary manner as long as S receives all garbled gates (e.g., a half of $\mathcal{G}_f$ from A and the other half from B). Besides equally splitting the work of circuit garbling between the parties, an alternative possibility is to let the weaker party (e.g., a mobile phone user) to do work sublinear in the circuit size. Let A be a weak client, who delegates as much work as possible to B. Then B generates the entire garbled circuit and sends it to S, while A will only need to create t_1 label pairs corresponding to her input, to be used in step 2 of the protocol. Upon completion of the result, A learns the output from B (i.e., there is no need for A to know labels for the output wires). Thus, the work and communication of the weaker client is only linear in the input and output sizes.

Security of this solution can be stated as follows, and the proof is deferred to the full version due to space constraints.

Theorem 1. *Protocol 1 fairly and securely evaluates function f in the presence of semi-honest A and B and malicious S.*

5.2 Semi-honest S, malicious A and B

To maintain efficiency of the previous solution by avoiding the cost of OT, we might want to preserve the high-level structure of the computation in the first solution. Now, however, because A and B can be malicious, neither of them can rely on the other party in garbling the circuit correctly. To address this, each of A and B may garble their own circuit for f, send it to S, and S will be in charge of evaluating both of them and performing a consistency check on the results (without learning the output). With this solution, A would create label pairs for her input bits/wires for both garbled circuits and communicate one set of pairs to B who uses them in constructing his circuit. What this achieves is that now A can directly send to S the labels corresponding to her input bits for circuit evaluation for both circuits. B performs identical operations. There is still no need to perform OT, but two security issues arise: (1) A and B must be forced to provide consistent inputs into both circuits and (2) regardless of whether the parties learn the output (e.g., whether the computation is aborted or not), a malicious party can learn one bit of information about the other party's input (by constructing a circuit that does not correspond to f) [41, 57]. While the first issue can be inexpensively addressed using the solution of [50] (which works in the presence of malicious users and semi-honest server), the second issue will still stand with this structure of the computation.

Instead of allowing for (1-bit) information leakage about private inputs, we change the way the computation takes place. If we now let the server garble the circuit and each of the remaining parties evaluate a copy of it, the need for OT (for both A and B's inputs) arises. We, however, were able to eliminate the use of OT for one of A and B and construct a solution that has about the same cost as a single two-party solution in the semi-honest model. At a high-level, it proceeds as follows: A creates garbled label pairs (ℓ_i^0, ℓ_i^1) for the wires corresponding to her inputs only and sends them to S. S uses the pairs to construct a garbled circuit for f and sends it to B. S and B engage in OT, at the end of which B learns labels corresponding to his input bits. Also, A sends to B the labels corresponding to her input bits, which allows B to evaluate the circuit. We note that because A may act maliciously, she might send to B incorrect labels, which will result in B's inability to evaluate the circuit. This, however, is equivalent to A aborting the protocol. In either case, neither A nor B learn any output and the solution achieves fairness. Similarly, if B does not perform circuit evaluation correctly, neither party learns the output.

The next issue that needs to to addressed is that of fairly learning the output. We note that S cannot simply send the label pairs for the output wires to A and B as this would allow B to learn the output and deny A of this knowledge. Instead, upon completion of garbled circuit evaluation, B sends the computed labels to A. With the help of S, A verifies that the labels A possesses are indeed valid labels for the output wires without learning the meaning of the output. Once A is satisfied, she notifies S who sends the label pairs to A and B, both of whom can interpret and learn the result. We note that malicious A can report failure to S even if verification of the validity of the output labels received from B was successful. Once again, this is equivalent to A aborting the protocol, in which case neither party learns the output and fairness is maintained.

Our solution is given as Protocol 2 and uses a hash function $H : \{0,1\}^* \to \{0,1\}^{\kappa}$ that we treat as a random oracle.

Input: A has private input x_1, B has private input x_2, and S has no private input.
Output: A and B learn $f(x_1, x_2)$, S learns nothing.
Protocol 2:

1. S chooses $\delta \xrightarrow{R} \{0,1\}^{\kappa-1}$, $k_1 \xrightarrow{R} \{0,1\}^{\kappa}$, $k_2 \xrightarrow{R} \{0,1\}^{\kappa}$ and sets $\Delta = \delta||1$. S sends Δ and k_1 to A.
2. S computes wire labels $\ell_i^0 = \mathsf{PRF}(k_1, i)$ for $i \in [1, t_1]$, $\ell_i^0 = \mathsf{PRF}(k_2, i - t_1)$ for $i \in [t_1 + 1, m]$, and sets $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [1, m]$. S then constructs garbled gates $\mathcal{G}_f$ and sends $\mathcal{G}_f$ to B.
3. S and B engage in t_2 instances of 1-out-of-2 OT, where S assumes the role of the sender and uses t_2 label pairs $(\ell_{t_1+i}^0, \ell_{t_1+i}^1)$ for $i \in [1, t_2]$ corresponding to B's input wires as its input and B assumes the role of the receiver and uses his t_2 input bits b_i as the input into the protocol. As the result of the interaction, B learns garbled labels $\ell_{t_1+i}^{b_i}$ for $i \in [1, t_2]$.
4. A computes labels $\ell_i^0 = \mathsf{PRF}(k_1, i)$ for $i \in [1, t_1]$ and sends to B $\ell_i^{b_i}$ for her input bits b_i, where $\ell_i^1 = \ell_i^0 \oplus \Delta$ for any $b_i = 1$.
5. After receiving the labels for his own and A's input, B evaluates the circuit, learns the output labels $\ell_i^{b_i}$ for $i \in [m - t_3 + 1, m]$ and sends them to A.
6. A requests from S output verification constructed as follows: For each output wire i, S computes $H(\ell_i^0)$, $H(\ell_i^1)$, randomly permutes the tuple, and sends it to A.
7. For each label ℓ_i received from B in step 5, A computes $H(\ell_i)$ and checks whether the computed value appear among $H(\ell_i^b)$, $H(\ell_i^{1-b})$ received from S in step 6. If the check succeeds for all output wires, A notifies S of success and aborts otherwise.
8. Upon receiving confirmation of success from A, S sends (ℓ_i^0, ℓ_i^1) for all output wires i to A and B, who recover the output.

We show security of this solution in a hybrid model where the parties are given access to a trusted entity computing OT. The proof can be found in Appendix C.

Theorem 2. *Protocol 2 fairly and securely evaluates function f in the presence of malicious A or B and semi-honest S in the hybrid model with ideal implementation of OT and where H is modeled as a random oracle.*

5.3 Semi-honest S, malicious A and B with input certification

We next consider an enhanced security setting in which malicious A and B are enforced to provide correct inputs in the computation. This enforcement is performed by requiring A and B to certify their inputs prior to protocol execution and prove the existence of certification on the inputs they enter.

The basic structure of our solution in this stronger security model remains the same as in Protocol 2, but we extend it with a novel mechanism for obliviously verifying correctness of the inputs. The intricate part of this problem is that signature schemes use public-key operations, while garbled circuit evaluation deals with randomly generated labels and symmetric key operations. In what follows, we describe the intuition behind our solution followed by more detailed explanation.

Suppose that the party whose inputs are to be verified participates in an OT protocol on her inputs as part of garbled circuit evaluation (i.e., the party is the circuit evaluator and acts as the receiver in the OT). Then if we use the variant of OT known as committed oblivious transfer (COT) (also called verifiable OT in some literature), the party will submit commitments to the bits of her input as part of OT computation and these commitments can be naturally tied to the values signed by a third party authority by means of ZKPKs (i.e., without revealing anything other than equality of the signed values and the values used in the commitments). Several COT schemes that we examined (such as in [45, 49]), however, had disadvantages in their performance and/or complex setup assumptions (such as requiring the sender and receiver to hold shares of the decryption key for a homomorphic public-key encryption scheme). We thus choose to integrate input certification directly with a conventional OT protocol by Naor and Pinkas [61].

Before we proceed with further description, we discuss the choice of the signature scheme and the way knowledge of a signature is proved. Between the main two candidates of signature schemes with protocols [16] and [17], we chose the one from [16] because it uses an RSA modulus. In application like ours, zero-knowledge statements are to be proved across different groups. This requires the use of statistically-hiding zero-knowledge proofs that connect two different groups through a setting in which the Strong RSA assumption (or, more generally, the difficulty of eth root extraction) holds [18, 27, 34]. Thus, the public key of the third party certification authority can be conveniently used as the common setup for other interaction between the prover and verifier. This has important implications on the use of such solutions in practice. (If multiple signatures are issued by multiple authorities, i.e., medical facilities in our application, one of the available public keys can be used to instantiate the common setup.)

Recall that in Protocol 2, B obtains the labels corresponding to his input from S via OT, while A knows all label pairs for her input wires and simply sends the appropriate labels to B. Now both of them have to prove to S that the inputs they enter in the protocol have been certified by a certain authority. For simplicity, in what follows we assume that all of A's and B's inputs are to be verified. (If this is not the case and only a subset of the inputs should be verified, the computation associated with input verification described below is simply omitted for some of the input bits.) Let us start with the verification mechanism for B, after which we treat the case of A.

B engages in the Naor-Pinkas OT in the role of the receiver. The details of the OT protocol are given in Appendix A. As part of OT, B forms two keys PK_0 and PK_1, where PK_σ is the key that will be used to recover m_σ. Thus, if we want to

enforce that σ corresponds to the bit for which B has a signature from a certification authority, B must prove that he knows the discrete logarithm of PK_σ where σ is the signed bit. More formally, the statement B has to prove in zero knowledge is $PK\{(\sigma,\beta) : \mathsf{Sig}(\sigma) \wedge y = \mathsf{Com}(\sigma) \wedge ((\sigma = 0 \wedge PK_0 = \hat{g}^\beta) \vee (\sigma = 1 \wedge PK_1 = \hat{g}^\beta))\}$. In other words, B has a signature on 0 and knows the discrete logarithm of PK_0 to the base $\hat{g}$ (i.e., constructed PK_0 as $\hat{g}^k$) or B has a signature on 1 and knows the discrete logarithm of PK_1 to the same base. Using a technically more precise PK statement for showing that σ is 0 or 1 would result in the PK statement above be re-written as $PK\{(\sigma,\alpha,\beta) : \mathsf{Sig}(\sigma) \wedge y = \mathsf{Com}(\sigma,\alpha) = g^\sigma h^\alpha \wedge ((y = h^\alpha \wedge PK_0 = \hat{g}^\beta) \vee (y/g = h^\alpha \wedge PK_1 = \hat{g}^\beta))\}$. We note that it is known how to realize this statement as a ZKPK as it uses only conjunction and disjunction of discrete logarithm-based sub-statements (see, e.g., [21]). Executing this ZKPK would allow S to verify B's input for a particular input wire if B has a signature on a bit. In practice, however, a signature is expected to be on messages from a larger space than $\{0,1\}$ and thus a single signature will need to be used to provide inputs for several input wires in the circuit. This can be accomplished by, in addition to using a commitment on the signed message, creating commitments to the individual bits and showing that they correspond to the binary representation of the signed message. Then the commitments to the bits of the message are linked to the keys generated in each instance of the OT. More formally, the ZKPK statement for a t-bit signed value would become:

$$PK\{(\sigma,\sigma_1,\ldots,\sigma_t,\alpha,\alpha_1,\ldots,\alpha_t) : \mathsf{Sig}(\sigma) \wedge y = g^\sigma h^\alpha \wedge y_1 = g^{\sigma_1} h^{\alpha_1} \wedge \ldots \wedge y_t = g^{\sigma_t} h^{\alpha_t} \wedge \sigma = \sum_{i=1}^{t} 2^{i-1}\sigma_i\} \quad (2)$$

$$PK\{(\sigma_i,\alpha_i,\beta_i) : y_i = g^{\sigma_i} h^{\alpha_i} \wedge ((y_i = h^{\alpha_i} \wedge PK_0^{(i)} = \hat{g}^{\beta_i}) \vee (y_i/g = h^{\alpha_i} \wedge PK_1^{(i)} = \hat{g}^{\beta_i}))\}. \quad (3)$$

Notation $PK_0^{(i)}$ and $PK_1^{(i)}$ denotes the public keys used in the ith instance of Naor-Pinkas OT. [21] shows how to prove that discrete logarithms satisfy a given linear equation.

Furthermore, it is likely that signatures will contain multiple messages (e.g., a genetic disease name and the outcome of its testing). In those cases, multiple messages from a single signature can be used as inputs into the garbled circuit or, depending on the function f, there might be other arrangements. For instance, one message can be used to provide inputs into the circuit and another be opened or partially open. It is not difficult to generalize equations 2 and 3 to cover such cases.

We now can proceed with the description of the mechanism for verifying A's inputs. Recall that for each bit i of her input, A has label pairs (ℓ_i^0, ℓ_i^1) and later sends to B the label $\ell_i^{b_i}$ corresponding to her input bit b_i. As before, consider first the case when A holds a signature on a single bit. To prove that the label $\ell_i^{b_i}$ sent to B corresponds to the bit for which she possesses a signature, we have A commit to the label $\ell_i^{b_i}$ and prove to S that either the commitment is to ℓ_i^0 and she has a signature on 0 or the commitment is to ℓ_i^1 and she has a signature on 1. Let the commitment be $c_i = \mathsf{Com}(\ell_i^{b_i}, r_i)$. Then if verification of the ZKPKs for each input bit was successful, S forwards each c_i to B together with the garbled circuit. Now when A sends her input label $\ell_i^{b_i}$ to B, she is also required to open the commitment c_i by sending r_i to B. B will proceed with circuit evaluation only if $c_i = g^{\hat{\ell}_i^{b_i}} h^{\hat{r}_i}$ for each bit i of A's input, where $\hat{\ell}_i^{b_i}$ and $\hat{r}_i$ are the values B received from A.

More formally, the statement A proves to S in ZK is $PK\{(\sigma,\alpha,\beta,\gamma) : \mathsf{Sig}(\sigma) \wedge y = g^\sigma h^\alpha \wedge z = g^\beta h^\gamma \wedge ((y = h^\alpha \wedge z/g^{\ell_i^0} = h^\gamma) \vee (y/g = h^\alpha \wedge z/g^{\ell_i^1} = h^\gamma))\}$. Similar to the case of B's input verification, this ZKPK can be generalized to use a single signature with multiple bits input into the circuit. More precisely, the statement in equation 2 remains unchanged, while the second statement becomes:

$$PK\{(\sigma_i,\alpha_i,\beta_i,\gamma_i) : y_i = g^{\sigma_i} h^{\alpha_i} \wedge z_i = g^{\beta_i} h^{\gamma_i} \wedge ((y_i = h^{\alpha_i} \wedge z_i/g^{\ell_i^0} = h^{\gamma_i}) \vee (y_i/g = h^{\alpha_i} \wedge z_i/g^{\ell_i^1} = h^{\gamma_i}))\}. \quad (4)$$

We summarize the overall solution as Protocol 3 in Appendix B. Its security can be stated as follows:

Theorem 3. *Protocol 3 fairly and securely evaluates function f in the presence of malicious A or B and semi-honest S in the hybrid model with ideal implementation of OT and where H is a hash function modeled as a random oracle and inputs of A and B are verified according to definition 2.*

Because the structure of the computation in Protocol 3 is the same as in Protocol 2 and primarily only ZKPKs have been added (that have corresponding simulators in the ideal model), we defer the proof to the full version of this work.

Before we conclude this section, let us comment on the possibility of using OT extensions in combination with certified inputs. First, notice that when only a subset of the input bits is to be verified, OT and OT extensions for the remaining input bits can be used as before. Second, if there is a computational benefit to using an OT extension instead of individual instances of OT, the benefit of an OT extension is not going to be as pronounced as in the regular case with no input certification. OT extensions allow the number of public key operations to be bounded by the security parameter and be independent of the number of input bits, while with certified inputs the number of public key operations is inevitably linear in the number of input bits being verified. For example, jumping ahead to experimental results, we see that computation in Protocol 3 in

Table 8 is 1 to 5 orders of magnitude larger for any given party than in Protocol 2 in Table 7 due to the use of certified inputs. This tells us that employing an OT extension in combination with certified inputs will have a negligible effect on the performance of Protocol 3. Furthermore, applicability of each individual OT extension mechanism to the case of certified inputs will likely need to be considered on a case by case basis and we leave this as a direction for future research. Publications that adapt OT extensions to new models (such as [51]) can be used as a starting point, but do not easily apply to our setting.

6 Private Genomic Computation

For all types of genomic computation we assume that A has information extracted from her genome, which she privately stores. Similarly, B stores data associated with his genome. A and B may enter some or all of their data into the computation and they may also compute a function of their individual data, which will be used as the input into the joint computation.

Ancestry test. This test would often be invoked when A and B already know to be related or have reasons to believe to be related. Under such circumstances, they are unlikely to try to cheat each other. For that reason, we use the solution with semi-honest A and B to realize this test. (Under the circumstances that this security model is not acceptable for some users A and B, they can always proceed with an alternative solution from this or other work, but we use the first protocol.) Because SNP-based tests are most general and can provide information about recent as well as distant ancestry, we build a circuit that takes a large number of SNPs from two individuals and counts the number of positions with the same values. The computed value is then compared to a number of thresholds to determine the closest generation in which the individuals have the same ancestor.

To compute the number of SNPs which are equal in the DNA of two individuals, the circuit first proceeds by XORing two binary input vectors from A and B (recall that the value of each SNP is a bit) and then counts the number of bits that differ in a hierarchical manner. That is, in the first round of additions, every two adjacent bits are added and the result is a 2-bit integer. In the second round of additions, every two adjacent results from the first round are added resulting in 3-bit sums. This process continues in $\lceil \log_2 t \rceil$ rounds of additions, where t is the size of A's and B's input, and the last round performs only a single addition. As mentioned earlier, the result can be interpreted by performing a number of comparisons at the end, but the cost of final comparisons is insignificant compared to the remaining size of the circuit.

Paternity test. We assess that the security setting with malicious users A and B is the most suitable for running paternity tests. That is, the participants may be inclined to tamper with the computation to influence the result of the computation. It is, however, difficult to learn the other party's genetic information by modifying one's input into the function. In particular, recall from equation 1 that the output of a paternity test is a single bit, which indicates whether the exact match was found. Then if a malicious participant engages in the computation with the same victim multiple times and modifies the input in the attempt to discover the victim's genomic data, the single bit output does not help the attacker to learn how his inputs are to be modified to be closer to the victim's input. The situation is different when the output of the computation reveals information about the distance between the inputs of A and B, but we do not consider such computation in this work. Thus, we do not use input certification for paternity tests.

This test would normally be run between an individual and a contested father of that individual according to the computation in equation 1. We thus implement the computation in equation 1 using a Boolean circuit. For each i, the circuit XORs the vectors $\langle x_{i,1}, x_{i,2}, x_{i,1}, x_{i,2} \rangle$ and $\langle x'_{i,1}, x'_{i,1}, x'_{i,2}, x'_{i,2} \rangle$ and compares each of the four value in the resulting vector to 0. The (in)equality to 0 testing is performed using $k-1$ OR gates, where k is the bitlength of all $x_{i,j}$'s and $x'_{i,j}$'s. Finally, we compute the AND of the results of the 4 equality tests, OR the resulting bits across i's, and output the complement of the computed bit.

Genetic compatibility test. When A and B want to perform a compatibility test, we assume that they want to evaluate the possibility of their children inheriting at least one recessive genetic disease. Thus, we assume that A and B agree on a list of genetic diseases to be included in the test (this list can be standard, e.g., suggested by S or a medical association). Because performing a test for a specific genetic disease is only meaningful if both parties wish to be tested for it, we assume that A and B can reconcile the differences in their lists.

To maximize privacy, we construct the function f to be as conservative as possible. In particular, given a list of genetic diseases L, A and B run a compatibility test for each disease $D \in L$, and if at least one test resulted in a positive outcome, the function will output 1, and otherwise it will output 0. That is, the function can be interpreted as producing 1 if A and B's children have a chance of inheriting the major variety for at least one of the tested diseases; and producing 0 means that their children will not inherit the major variety for any of the diseases in L. Evaluating this function can be viewed as the first step in A and B's interaction. If the output was 1, they may jointly decide to run more specific computation to determine the responsible disease or diseases themselves.

The above means that for each $D \in L$, A can locally run the test to determine whether she is a carrier of D. B performs the same test on his data. Thus, A's and B's input into f consists of $|L|$ bits each and the result is 1 iff $\exists i$ such that A's and B's ith input bits are both 1. This computation can be realized as a simple circuit consisting of $|L|$ AND and $|L|-1$ OR gates.

Next, notice is that it is easy for malicious A or B to learn sensitive information about the other party by using certain inputs. That is, if a malicious user sets all his input bits to 1, he will be able to learn whether the other party is a carrier of least one disease in L. This poses substantial privacy concerns, particularly for matchmaking services that routinely run genetic compatibility tests between many individuals. Thus, we require that A and B certify the results of testing for each genetic disease on the list (e.g., by a medical facility) and enter certified inputs into the computation. (Note that the medical facility that performs sequencing can also certify the test results; alternatively, the medical facility performing test certification will require genome certification from the facility that performed sequencing.) This means that the server-aided solution with certified inputs will be used for secure computation.

For each disease $D \in L$, the signature will need to include the name of the disease D and the test outcome σ, which we assume is a bit. Then if we target efficient the computation, the disease names will not be input into the circuit, but instead S will verify that A's signature used for a particular input wire includes the same disease name as B's signature used for an equivalent input wire. A simple way to achieve this is to reveal list L to S and reveal the name of the disease including in each signature (without revealing the signature itself). If we assume that each issued signature is on the tuple (D, σ), i.e., the signature was produced as $v^e \equiv a_1^D a_2^\sigma b^s c$, all that is needed is to adjust the value used in the ZKPK of the signature by $\frac{1}{a_2^D}$ by both the sender and the verifier for each $D \in L$ (we refer the reader to [16] for details). S will need to check that all conditions appear in the same order among A's and B's inputs (i.e., the sequences of diseases are identical) before proceeding with the rest of the protocol. Revealing the set of diseases used in the compatibility test would not constitute violation of privacy if such a set of conditions is standard or suggested by S itself.

When, however, the parties compose a custom set of genetic diseases for their genetic compatibility testing and would like to keep the set private, they may be unwilling reveal the set of diseases to S. We propose that the parties instead prove that they are providing results for the same conditions without revealing the conditions themselves to the server. The difficulty in doing so arises from the fact that S interacts independently with A and B (possibly at non-overlapping times) and A and B are not proving any joint statements together. Our idea of proving that inputs of A and B correspond to the same sequence of diseases consists of forming a sequence of commitments to the diseases in L, the openings of which are known to both A and B. That is, A and B jointly generate a commitment to each disease using shared randomness and used those commitments at the time of proving that their inputs have been certified. Then if A supplies commitments $\mathsf{Com}_1, \ldots, \mathsf{Com}_t$ and proves that the committed values correspond to the diseases in her signatures, S will check that B supplies the same sequence of commitments and also proves that the committed values are equal to the diseases in the signatures he possesses. This will ensure that A and B supply input bits for the same sequence of diseases. To jointly produce the commitments, we have both A and B contribute their own randomness and the resulting commitment will be a function of A's and B's contribution. It can proceed as follows (recall that A and B can be malicious):

1. A chooses a random r_A and sends to B $c_A = \mathsf{Com}(r_A, z)$.
2. B chooses a random r_B and sends to A $c_B = \mathsf{Com}(r_B, z')$.
3. A and B open their commitments by exchanging (r_A, z) and (r_B, z') and verify that they match c_A and c_B, resp.
4. They form joint randomness as $r = r_A \oplus r_B$ and use it to construct commitment $\mathsf{Com}(D, r)$.

Then the (high-level) statement that A and B prove about their inputs is $PK\{(\alpha, \sigma) : \mathsf{Sig}(\alpha, \sigma) \wedge y_1 = \mathsf{Com}(\alpha) \wedge y_2 = \mathsf{Com}(\sigma)\}$ using y_1 shared between A and B, while the remaining portion is specific to A and B as detailed in section 5.3.

7 Performance Evaluation

In this section, we report on the results of our implementation. The implementation was written in C/C++ using Miracl library [5] for large number arithmetic and JustGarble library [4] for garbled circuit implementation. We provide experimental results for ancestry, paternity, and compatibility tests implemented as described in section 6 as well as additional functions, on all of which we further elaborate below. The security parameters for symmetric key cryptography and statistical security were set to 128. The security parameter for public key cryptography (for both RSA modulus and discrete logarithm setting) was set to 1536. Additionally, the security parameter for the group size in the discrete logarithm setting was set to 192. All tests (for A, B, and S) were run on a quad-core 3.2GHz machine with Intel i5-3470 processor running Red Hat Linux 2.6.32 in a single core. Note that in practice S is expected to have more powerful hardware and the runtimes can be significantly reduced by utilizing more cores. All experiments were run 5 times, and the mean value is reported.

To provide additional insights into which protocol component is main performance bottleneck, we separately report computation times for different parts of each solution (e.g., garbled circuit evaluation, OT, etc.). Furthermore, we sepa-

Table 2. Performance of ancestry test without/with half-gates.

Party	Garbled circuit		Communication	
	garble (offline)	eval (online)	sent	received
A	1.8ms	–	2MB	0MB
B	19.8/18.4ms	–	8/6MB	0MB
S	–	12.5/15.9ms	0MB	10/8MB

rately list the times for offline and online computation, where, as in other publications, offline computation refers to all operations that can be performed before the inputs become available. Lastly, because the speed of communication channels can greatly vary, we separately report the size of communication for each party and communication time is not included in the runtimes. In several cases overlaying computation with communication is possible (e.g., S can perform OT computation and simultaneously transmit the garbled circuit) and the overall runtime does not need to be the sum of computation and communication time. We first discuss ancestry, paternity, and compatibility tests in their respective settings and then proceed with evaluating additional functions in all three settings.

7.1 Ancestry test

Recall that the ancestry test is implemented in the setting where A and B are semi-honest, but S can be malicious. We ran this test using 2^{17} SNPs as the input for A and B. The resulting circuit used 655,304 XOR gates and 131,072 non-XOR gates. The computation time and communication size are given in Table 2. We used the original JustGarble implementation as well as implement a variant with the recent half-gates optimization [63], which reduces bandwidth associated with transmitting garbled circuits.[1] Both variants are listed in Table 2. In the context of this work, the half-gates optimization has the largest impact on the performance of the first protocol, as in the remaining protocols other components of SFE are likely to dominate the overall time.

The implementation assumes that A only creates labels for her input wires and communicates 2^{17} labels to B. B performs the rest of the garbling work and interacts with S. As expected, the time for circuit garbling and evaluation is small, but the size of communication is fairly large because of the large input size and consecutively circuit size. Nevertheless, we consider the runtimes very small for the computation of this size.

1 In both the original and half-gates implementations, garbling a non-free gate involves calling AES on 4 blocks, while evaluation of a half gate calls AES on 2 blocks and on 1 block in the original implementation. Any deviations in the run time from these expectations are due to non-cryptographic operations.

To provide insights into performance gains of our solution compared to the regular two-party computation in the semi-honest setting, we additionally implement the garbled circuit-based approach in the presence of semi-honest A and B only. In addition to circuit garbling and evaluation, this also requires the use of OT, which we implement using a recent optimized OT extension construction from [7] (including optimizations specific to Yao's garbled circuit evaluation). As in [7], we use Naor-Pinkas OT for 128 base OTs [61]. The results are given in Table 3. Compared to the server-aided setting, computation is higher by at least two orders of magnitude for each party and communication is noticeably increased as well.

7.2 Paternity test

Next, we look at the paternity test, implemented as described in section 6 in the presence of malicious A and B and semi-honest S. The inputs for both A and B consisted of 13 2-element sets, where each element is 9 bits long. We use OT extension from [7] with 128 Naor-Pinkas base OTs. The circuit consisted of 468 XOR and 467 non-XOR gates. The results of this experiment are reported in Table 4. The computation for output verification is reported only as part of total time. Not surprisingly, the cost of OT dominates the overall runtime, but for A the overhead is negligible (the cost of generating input labels and verifying the output labels returned by B). Thus, it is well-suited for settings when one user is very constrained.

Compared to two-party computation in the presence of malicious participants, our solution reduces both computation and communication for the participants by at least two orders of magnitude. This is because practical constructions rely on cut-and-choose (and other) techniques to ensure that the party who garbles the circuit in unable to learn unauthorized information about the other participant's input. Recent results such as [6, 56, 58] require the circuit generator to garble on the order of 125 circuits for cheating probability of at most 2^{-40}, some of which are checked (i.e., re-generated) by the circuit evaluator, while the remaining circuits are evaluated. Thus, the work of each of A and B will have to increase by at least two orders of magnitude just for circuit garbling and evaluation, not counting other techniques that deter a number of known attacks and result in increasing the input size and introducing expensive public key operations. A notable exception to the above is the work of Lindell [54] that reduces the number of circuits to 40 for the same cheating probability. The construction, however, results in savings only for circuits of large size as it introduces a large number of additional public key operations. Thus, for paternity tests using constructions with a larger number of circuits is very likely to be faster in practice, which results in a drastic difference between our solution and regular

Table 3. Performance of ancestry test without server without/with half-gates.

Party	Garbled circuit		OT		Total time		Comm	
	garble	eval	offline	online	offline	online	sent	received
A	21.6/20.2ms	–	195.2ms	1983ms	216.8/215.4ms	1983ms	10.02/8.02MB	2.03MB
B	–	12.5/15.9ms	2003ms	218.6ms	2003ms	231.1/234.5ms	2.03MB	10.02/8.02MB

Table 4. Performance of paternity test (no half-gates); work is in ms, communication is in KB.

Party	GC		OT		Total time		Comm	
	garble	eval	offline	online	offline	online	sent	recvd
A	0.003	–	–	–	0.003	–	3.7	0.06
B	–	0.01	515.5	201.7	515.5	201.7	31.67	56.88
S	0.03	–	196.1	260.9	196.1	260.9	53.32	31.66

two-party protocol with malicious participants. This difference in performance can be explained by the fact that in our setting one party is known not to deviate from the protocol allowing for a more efficient solution. We also provide a comparison to recent general three-party constructions in section 7.4.

Baldi et al. [12] also provide a private paternity test in the two-party setting (between a client and a server). It uses a different computation based on Restriction Fragment Length Polymorphisms (RFLPs) and relies on private set intersection as a cryptographic building block. Both offline and online times for the client and the server are 3.4 ms and the communication size is 3KB for the client and 3.5KB for the server when the test is performed with 25 markers. All times and communication sizes double when the test is run with 50 markers. While the runtimes we report are higher, the implementation of [12] did not consider malicious participants. If protection against malicious A and B in our solution is removed, the work for all parties reduces to well below 0.1 millisecond and communication becomes a couple of KBs.

7.3 Genetic compatibility test

The last genetic compatibility test is run in the setting where A and B are malicious and their inputs must be certified. We choose the variant of the solution that reveals the list of diseases L to the server (i.e., a standard list is used). We implement the signature scheme, OT, and ZKPKs as described earlier. All ZKPKs are non-interactive using the Fiat-Shamir heuristic [33]. We used $|L| = 10$ and thus A and B provide 10 input bits into the circuit accompanied by 10 signatures. The circuit consisted of only 19 non-XOR gates. The performance of the test is given in Table 5. We divide all ZKPKs into a proof of signature possession (together with a commitment), denoted by "Sign PK" in the table, and the remaining ZK proofs, denoted by "Other PK." As it is clear from the table, input certification contributes most of the solution's overhead, but it is still on the order of 1–3 seconds for all parties.

As mentioned earlier, we are not aware of general results that achieve input certification for comparison. However, the comparison to general two-party computation in the presence of malicious parties or server-aided two-party computation from sections 7.2 and 7.4 applies here as well.

Baldi et al. [12] also build a solution and report on the performance of genetic compatibility test. In [12], testing for presence of a genetic disease that client carries in the server genome consists of the client providing the disease fingerprint in the form of $(nucleotide, location)$ pairs (which is equivalent to a SNP) and both parties searching whether the disease fingerprint also appears in the server's DNA. This requires scanning over the entire genome, which our solution avoids. As a result, the solution of [12] incurs substantial offline overhead for the server (67 minutes) and large communication size (around 4GB) even for semi-honest participants. The solution utilizes authorized private set intersection, which allows inputs of one party (as opposed to both in our work) to be verified. Compared to [12], in our framework, testing for a single disease requires a fraction of a second for each party with malicious A and B, where inputs of both of them are certified. The computation is greatly simplified because the list of diseases is assumed to be known by both users. When this is the case, the cost of input certification greatly dominates the overall time.

7.4 Additional functions

To better understand performance of our solutions for a variety of functions, we next present the results of evaluating a number of functionalities in all three settings put forward in this work. We evaluate AES (standard test), hamming distance (addition-heavy), matrix multiplication (multiplication-heavy), and edit distance (comparison-heavy) as representative functions used in related literature. Tables 6, 7, and 8 provide performance results for protocols 1, 2, and 3, respectively, (no half gates) and table 10 in Appendix D reports on the performance of protocol 1 with half-gates (recall that this optimization has the most impact on the first protocol). The inputs are either n-bit strings, $n \times n$ matrices of 32-bit integers, or n-bit strings of 8-bit characters for the choice of n listed in the tables.

Table 5. Performance of compatibility test (no half-gates).

Party	Garbled circuit		OT		Sign PK		Other PK		Total time		Comm	
	garble	eval	offline	online	offline	online	offline	online	offline	online	sent	received
A	0ms	–	–	–	1170ms	42.1ms	616.8ms	20.6ms	1790ms	62.7ms	34.35KB	0.06KB
B	–	0.001ms	15.4ms	14.6ms	1170ms	42.1ms	282.4ms	15.7ms	1470ms	72.4ms	36.41KB	2.98KB
S	0.003ms	–	29.3ms	15.2ms	0	2060ms	0ms	756ms	29.3ms	2830ms	2.87KB	70.59KB

All of [23, 24, 38, 47, 60] provide server-aided secure computation schemes for general functionalities that could be compared to our constructions. The solution of [38] has the closest setting to our work (protocol 2) that assumes two malicious users and a semi-honest server. While no implementation was provided in [38], the construction of [38] would require A and B to verify on the order of $O(\kappa n)$ signatures and engage in $O(\kappa n)$ OTs (where n is the number of input bits), which translates into tens of thousands of public key operations even for simple functions and significantly larger volume of communication than in protocol 2. The server's work in [38] is larger than in our solution as well. The way the server is used, however, is more constrained than in our work.

Whitewash [23] improves on the result of Carter et al. [24] and thus we include performance comparison only for the former. In addition, Mood et al. [60] improves on the result of [24] by allowing a garbled value from one circuit to be transformed to a garbled input of another circuit, thus allowing for computation to be performed in stages and reusing values from one circuit in a different circuit. This allows for savings associated with input transfer and validation. However, for the functions we report in this section (such as matrix multiplication and edit distance), [60] did not show observable improvement in runtime per circuit when multiple circuits are executed instead of a single circuit. Thus, we do not provide a direct comparison of our results with the performance of [60]. The techniques of [60] appear to be most effective for circuits with a high ratio of input bits to the circuit size.

Because of the limitations of the underlying PCF compiler [53] on which Whitewash builds, we were able run Whitewash only on the circuits included with the tool. In particular, we were unable to run AES and edit distance experiments, as well as matrix multiplication for 4×4 matrices. The details of Whitewash performance on the same setup as in our other experiments are provided in table 9. The security setting of Whitewash is the closest to our protocol 2. If we then compare the overhead in tables 7 and 9, we see that computation time is at least 3 orders of magnitude less in protocol 2 for all parties (phone in Whitewash corresponds to our party A) and communication is 2 to 3 orders of magnitude less in protocol 2. Our savings are possible because the security model of [60] is more challenging (where any participating party can act maliciously). Furthermore, the goal of [60] was not to optimize the overall work, but rather lower the overhead of the weak party.

Kamara et al. [47] uses server-aided computation with any number of participants in a somewhat different security model. The authors of [47] we unable to share their implementation with us and thus we compare performance with the numbers reported in [47]. In the 2-party plus server setting, [47] reports simplified AES performance (without key expansion) on the order of 40 seconds and hundreds of MBs in communication, while we obtain about 1 second total time and communication less than 0.5MB. For 50-character edit distance, [47] reports 240 seconds runtime with over 1GB communication, while we achieve on the order of 1 second runtime with 24MB communication for 64-character strings. Once again, the performance gap can be justified by the differences in the security model.

8 Conclusions

This work is motivated by the need to protect sensitive genomic data when it is used in computation, especially in voluntary non-health related computation. Because computation over one's genome often happens in server-facilitated settings, we study server-aided secure two-party computation in a number of security settings. One of such security settings assumes that users A and B may act arbitrarily and, in addition to requiring security in the presence of malicious users, we also enforce that A and B enter their true inputs based on third party certification. We are not aware of any prior work that combines input certification with general secure multi-party computation based on Yao's garbled circuits. We develop general solutions in our server-aided framework. Despite their generality, they lead to efficient implementations of genetic tests. In particular, we design and implement genetic paternity, compatibility, and common ancestry tests, all of which run in a matter of seconds or less and favorably compare with the state of the art.

9 Acknowledgments

The authors thank Aaron Steele for useful discussions of genomic tests at early stages of this work, Dagstuhl seminar on

Table 6. Performance of protocol 1 (no half-gates); work is in ms.

Function	Input size	Party	Computation			Communication		
			offl.	onl.	total	sent	recvd	total
AES	128	A	0.02	–	0.02	2KB	0.01KB	2KB
		B	0.46	–	0.46	272KB	0.11KB	272KB
		S	–	0.15	0.15	0.11KB	274KB	274KB
Hamming distance (bits)	2^{12}	A	0.06	–	0.06	64KB	0	64KB
		B	0.34	–	0.34	256KB	0.2KB	256KB
		S	–	0.17	0.17	0.19KB	320KB	320KB
	2^{13}	A	0.11	–	0.11	128KB	0	128KB
		B	0.73	–	0.73	507KB	0.2KB	507KB
		S	–	0.38	0.38	0.2KB	635KB	636KB
	2^{14}	A	0.22	–	0.22	256KB	0	256KB
		B	1.83	–	1.83	1.0MB	0KB	1.0MB
		S	–	1.1	1.1	0.4KB	1.25MB	1.25MB
Matrix multiplication ($n \times n$ ints)	4	A	0.01	–	0.01	8KB	0.06KB	8KB
		B	14.5	–	14.5	9.0MB	8KB	9.0MB
		S	–	7.49	7.49	8KB	9.0MB	9.0MB
	8	A	0.03	–	0.03	32KB	0.25KB	32KB
		B	116	–	116	72MB	32KB	72MB
		S	–	59.7	59.7	32KB	72MB	72MB
	16	A	0.11	–	0.11	128KB	1KB	129KB
		B	926	–	926	576MB	128KB	576MB
		S	–	476	476	128KB	576MB	576MB
Edit distance (chars)	32	A	0.00	–	0.00	4KB	0	4KB
		B	9.52	–	9.52	61MB	0.1KB	61MB
		S	–	3.33	3.33	0.08KB	61MB	61MB
	64	A	0.01	–	0.01	8KB	0	8KB
		B	38.1	–	38.1	24MB	0.9KB	24MB
		S	–	13.3	13.3	0.9KB	24MB	24MB
	128	A	0.01	–	0.01	16KB	0	16KB
		B	153	–	153	97.5MB	0.11KB	97.5MB
		S	–	53.5	53.5	0.11KB	97.4MB	97.4MB

Table 7. Performance of protocol 2 (no half-gates); work is in ms unless noted otherwise.

Function	Input size	Party	Computation			Communication		
			offl.	onl.	total	sent	recvd	total
AES	128	A	0.02	–	0.02	2KB	8KB	10KB
		B	201	191	392	30KB	300KB	330KB
		S	382	205	587	304KB	28KB	332KB
Hamming distance (bits)	2^{12}	A	0.06	–	0.06	64KB	0.75KB	65KB
		B	200	381	581	92.2KB	408KB	500KB
		S	561	205	766	345KB	92KB	437KB
	2^{13}	A	0.11	–	0.1	128KB	0.81KB	129KB
		B	200	429	629	156KB	787KB	943KB
		S	604	206	810	660KB	156KB	437KB
	2^{14}	A	0.22	–	0.22	256KB	0.88KB	257KB
		B	201	537	738	284KB	1.45MB	1.72MB
		S	701	208	909	1.27MB	369KB	1.63MB
Matrix multiplication ($n \times n$ ints)	4	A	0.01	–	0.01	8KB	32KB	40KB
		B	214	335	549	42KB	9.18MB	9.18MB
		S	523	212	735	9.08MB	36KB	9.11MB
	8	A	0.03	–	0.03	32KB	128KB	160KB
		B	316	355	671	92KB	72.2MB	72.3MB
		S	540	265	805	72.2MB	60KB	72.3MB
	16	A	0.11	–	0.1	128KB	512KB	640KB
		B	1.1s	429	1.6s	184KB	577MB	577MB
		S	604	682	1.3s	577MB	156KB	577MB
Edit distance (chars)	32	A	0.00	–	0.00	4KB	0.31KB	4.31KB
		B	209	333	542	32KB	6.13MB	6.13MB
		S	522	208	730	6.13MB	32KB	6.13MB
	64	A	0.01	–	0.01	8KB	0.37KB	8.4KB
		B	237	335	572	36.1KB	24.4MB	24.4MB
		S	523	218	741	24.4MB	36KB	24.4MB
	128	A	0.01	–	0.01	16KB	0.44KB	16.4KB
		B	352	342	694	44.1KB	97.6MB	97.6MB
		S	528	258	786	97.6MB	44KB	97.6MB

Genomic Privacy for motivating this work, and anonymous reviewers for their valuable feedback. The authors are also greatful to Henry Carter and Benjamin Mood for their help with running Whitewash and PCF compiler experiments. Portions of this work were supported by grants CNS-1223699 and CNS-1319090 from the US National Science Foundation and grant AFOSR-FA9550-13-1-0066 from the US Air Force Office of Scientific Research. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] 23andMe – Genetic Testing for Ancestry; DNA Test. http://www.23andme.com.

[2] Genealogy, Family Trees & Family History Records at Ancestry.com. http://www.ancestry.com.

[3] GenePartner.com – DNA matching: Love is no coincidence. http://www.genepartner.com.

[4] The JustGarble library. http://cseweb.ucsd.edu/groups/justgarble/.

[5] The Miracl library. http://http://www.certivox.com/miracl/.

[6] a. shelat and C. h. Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT*, 2011.

[7] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*, 2013.

[8] E. Ayday, J. L. Raisaro, and J. Hubaux. Personal use of genomic data: Privacy vs. storage cost. In *IEEE Global Communications Conference*, pages 2723–2729, 2013.

[9] E. Ayday, J. L. Raisaro, and J.-P. Hubaux. Privacy-enhancing technology for medical tests using genomic data. Technical Report EPFL-REPORT-182897, EPFL, 2012.

[10] E. Ayday, J. L. Raisaro, J.-P. Hubaux, and J. Rougemont. Protecting and evaluating genomic privacy in medical tests and personalized medicine. In *WPES*, pages 95–106, 2013.

[11] E. Ayday, J. L. Raisaro, P. McLaren, J. Fellay, and J.-P. Hubaux. Privacy-preserving computation of disease risk by using genomic, clinical, and environmental data. In *HealthTech*, 2013.

Table 8. Performance of protocol 3 (no half-gates); work is in sec unless noted otherwise.

Function	Input size	Party	Computation			Communication		
			offl.	onl.	total	sent	recvd	total
AES	128	A	8.07	0.28	8.35	145KB	8KB	153KB
		B	3.99	0.4	4.39	173KB	300KB	473KB
		S	0.38	10.2	10.6	304KB	313KB	617KB
Hamming distance (bits)	2^{12}	A	253	8.48	261	4.43MB	0.75KB	4.43MB
		B	116	6.85	123	4.46MB	408KB	4.86MB
		S	0.56	311	312	345KB	8.83MB	9.17MB
	2^{13}	A	506	16.9	523	8.87MB	0.81KB	8.87MB
		B	233	13.4	246	8.89MB	879KB	9.75MB
		S	0.60	622	623	660KB	17.6MB	18.3MB
	2^{14}	A	0.02	33.9	33.9	17.8MB	0.88KB	17.8MB
		B	465	26.4	491	17.7MB	1.44MB	19.2MB
		S	0.70	21m	21m	1.27MB	35.4MB	36.7MB
Matrix multiplication ($n \times n$ ints)	4	A	31.7	1.07	32.8	569KB	32KB	601KB
		B	14.8	1.15	15.9	603KB	9.18MB	9.77MB
		S	0.52	39.3	39.8	9.08MB	1.14MB	10.2MB
	8	A	127	4.24	131	2.22MB	128KB	2.32MB
		B	58.5	3.60	62.1	2.28MB	72.2MB	74.5MB
		S	0.54	156	157	72.2MB	4.43MB	76.4MB
	16	A	506	16.9	523	8.87MB	512KB	9.37MB
		B	234	13.4	247	8.92MB	577MB	586MB
		S	0.60	622	623	577MB	17.6MB	594MB
Edit distance (chars)	32	A	16.0	0.54	16.5	286KB	0.31KB	286KB
		B	7.61	0.75	8.36	314KB	6.13MB	6.44MB
		S	0.52	19.9	20.4	6.13MB	596KB	6.72MB
	64	A	31.7	1.07	32.8	569KB	0.37KB	569KB
		B	14.9	1.15	16.0	597KB	24.4MB	25.0MB
		S	0.52	39.3	39.8	24.4MB	1.14MB	25.5MB
	128	A	63.3	2.12	65.4	1.11MB	0.44KB	1.11MB
		B	29.4	1.96	31.4	40.2MB	97.5MB	138MB
		S	0.53	78.0	78.5	97.5MB	2.24MB	99.7MB

Table 9. Performance of Whitewash [23].

Function	Input size	Party	Total comp time	Total comm size	Total exec time
Hamming distance (bits)	2^{12}	Eval	861s	450MB	1077s
		Gen	1008s	450MB	
		Phone	3.46s	60.2MB	
	2^{13}	Eval	1716s	901MB	2117s
		Gen	1999s	901MB	
		Phone	6.86s	121MB	
	2^{14}	Eval	3362s	1.54GB	4128s
		Gen	3918s	1.54GB	
		Phone	13.7s	241MB	
Matrix multiplication ($n \times n$ ints)	3	Eval	97.7s	230MB	176s
		Gen	124s	230MB	
		Phone	0.29s	4.26MB	
	5	Eval	300s	1.00GB	520s
		Gen	357s	1.00GB	
		Phone	0.71s	11.8MB	
	8	Eval	922s	4.01GB	1615s
		Gen	1007s	4.01GB	
		Phone	1.76s	30.2MB	
	16	Eval	5613s	31.5GB	10716s
		Gen	5991s	31.5GB	
		Phone	6.87s	121MB	

[12] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik. Countering GATTACA: Efficient and secure testing of fully-sequenced human genomes. In *CCS*, pages 691–702, 2011.

[13] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium of Security and Privacy*, pages 478–492, 2013.

[14] M. Beye, Z. Erkin, and R. Lagendijk. Efficient privacy preserving k-means clustering in a three-party setting. In *WIFS*, pages 1–6, 2011.

[15] F. Bruekers, S. Katzenbeisser, K. Kursawe, and P. Tuyls. Privacy-preserving matching of DNA profiles. IACR Cryptology ePrint Archive Report 2008/203, 2008.

[16] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *SCN*, pages 268–289, 2002.

[17] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, pages 56–72, 2004.

[18] J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In *CRYPTO*, 1999.

[19] J. Camenisch, D. Sommer, and R. Zimmermann. A general certification framework with applications to privacy-enhancing certificate infrastructures. In *Security and Privacy in Dynamic Environments*, pages 25–37, 2006.

[20] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO*, 1997.

[21] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical report, Institute for Theoretical Computer Science, ETH Zurich, 1997.

[22] J. Camenisch and G. Zaverucha. Private intersection of certified sets. In *FC*, pages 108–127, 2009.

[23] H. Carter, C. Lever, and P. Traynor. Whitewash: Outsourcing garbled circuit generation for mobile devices. In *ACSAC*, pages 266–275, 2014.

[24] H. Carter, B. Mood, P. Traynor, and K. Butler. Secure outsourced garbled circuit evaluation for mobile devices. In *USENIX Security Symposium*, 2013.

[25] H. Carter, B. Mood, P. Traynor, and K. Butler. Outsourcing secure two-party computation as a black box. In *CANS*, pages 214–222, 2015.

[26] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *STOC*, pages 573–588, 1986.

[27] I. Damgard and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT*, pages 125–142, 2002.

[28] E. De Cristofaro, S. Faber, P. Gasti, and G. Tsudik. GenoDroid: Are privacy-preserving genomic tests ready for prime time? In *WPES*, pages 97–107, 2012.

[29] E. De Cristofaro, S. Faber, and G. Tsudik. Secure genomic testing with size- and position-hiding private substring matching. In *WPES*, pages 107–118, 2012.

[30] E. De Cristofaro, S. Faber, and G. Tsudik. Secure genomic testing with size- and position-hiding private substring matching. In *WPES*, pages 107–118, 2013.

[31] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security (FC)*, pages 143–159, 2010.

[32] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation. In *STOC*, pages 554–563, 1994.

[33] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature scheme. In *CRYPTO*, pages 186–194, 1986.

[34] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, pages 16–30, 1997.

[35] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[36] S. Goldwasser, Y. Kalai, and G. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[37] D. He, N. Furlotte, F. Hormozdiari, J. Joo, A. Wadia, R. Ostrovsky, A. Sahai, and E. Eskin. Identifying genetic relatives without compromising privacy. *Genome Research*, 24:664–672, 2014.

[38] A. Herzberg and H. Shulman. Oblivious and fair server-aided two-party computation. In *ARES*, pages 75–84, 2012.

[39] A. Herzberg and H. Shulman. Oblivious and fair server-aided two-party computation. *Information Security Technical Report*, (17):210–226, 2013.

[40] F. Hormozdiari, J. Joo, A. Wadia, F. Guan, R. Ostrovsky, A. Sahai, and E. Eskin. Privacy preserving protocol for detecting genetic relatives using rare variants. In *ISMB*, pages 204–2011, 2014.

[41] Y. Huang, J. Katz, and D. Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium of Security and Privacy*, 2012.

[42] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.

[43] Y. Ishai, R. Kumaresan, E. Kushilevitz, and A. Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *CRYPTO*, pages 359–378, 2015.

[44] T. Jakobsen, J. Nielsen, and C. Orlandi. A framework for outsourcing of secure computation. In *ACM Workshop on Cloud Computing Security (CCSW)*, pages 81–92, 2014.

[45] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT*, pages 97–114, 2007.

[46] S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. IACR Cryptology ePrint Archive Report 2011/272, 2011.

[47] S. Kamara, P. Mohassel, and B. Riva. Salus: A system for server-aided secure function evaluation. In *CCS*, pages 797–808, 2012.

[48] J. Katz and L. Malka. Secure text processing with applications to private DNA matching. In *CCS*, pages 485–492, 2010.

[49] M. Kiraz, T. Schoenmakers, and J. Villegas. Efficient committed oblivious transfer of bit strings. In *Information Security Conference (ISC)*, pages 130–144, 2007.

[50] V. Kolesnikov, R. Kumaresan, and A. Shikfa. Efficient verification of input consistency in server-assisted secure function evaluation. In *CANS*, pages 201–217, 2012.

[51] V. Kolesnikov and A. Malozemoff. Public verifiability in the covert model (almost) for free. In *ASIACRYPT*, 2015.

[52] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, pages 486–498, 2008.

[53] B. Kreuter, a. shelat, B. Mood, and K. Butler. PCF: A portable circuit format for scalable two-party secure computation. In *USENIX Security Symposium*, 2013.

[54] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, 2013.

[55] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[56] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology*, 25(4):680–722, 2012.

[57] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography*, pages 458–73, 2006.

[58] P. Mohassel and B. Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In *CRYPTO*, pages 36–53, 2013.

[59] P. Mohassel, M. Rosulek, and Y. Zhang. Fast and secure three-party computation: The garbled circuit approach. In *CCS*, pages 591–602, 2015.

[60] B. Mood, D. Gupta, K. Butler, and J. Feigenbaum. Reuse it or lose it: More efficient secure computation through reuse of encrypted values. In *CCS*, pages 582–596, 2014.

[61] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.

[62] J. Nielsen, P. Nordholt, C. Orlandi, and S. Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700, 2012.

[63] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, pages 220–250, 2015.

A Additional Background

Genomic background. Genomes represent complete hereditary information of an individual. Information extracted from one's genome can take different forms. One type is called Single Nucleotide Polymorphisms (SNPs), each of which corresponds to a well known variation in a single nucleotide (a nucleotide can be viewed as a simple unit represented by a letter A, C, G, or T). Because SNP mutations are often associated with how one develops diseases and responds to treatments, they are commonly used in genetic disease and disorder testing. The same set of SNPs (i.e., nucleotides in the same positions) would be extracted for each individual, but the values associated with each SNP differ from one individual to another. Normally each SNP is referenced by a specific index and its value in a individual is represented as a bit, while representations consisting of 3 values 0, 1, 2 are used as well.

Input: Sender S has two strings m_0 and m_1, receiver R has a bit σ. Common input consists of prime p, generator $\hat{g}$ of subgroup of $\mathbb{Z}_p^*$ of prime order q, and a random element C from the group generated by $\hat{g}$ (chosen by S).
Output: R learns m_σ and S learns nothing.
OT Protocol:

1. S chooses random $r \in \mathbb{Z}_q$ and computes C^r and $\hat{g}^r$.
2. R chooses $k \in \mathbb{Z}_q^*$, sets public keys $PK_\sigma = \hat{g}^k$ and $PK_{1-\sigma} = C/PK_\sigma$, and sends PK_0 to S.
3. After receiving PK_0, S computes $(PK_0)^r$ and $(PK_1)^r = C^r/(PK_0)^r$. S sends to R $\hat{g}^r$ and two encryptions $H((PK_0)^r, 0) \oplus m_0$ and $H((PK_1)^r, 1) \oplus m_1$, where H is a hash function (modeled as a random oracle).
4. R computes $H((\hat{g}^r)^k) = H((PK_\sigma)^r)$ and uses it to recover m_σ.

Fig. 1. 1-out-of-2 Oblivious Transfer of [61].

Another type of data extracted from a genome is based on Short Tandem Repeats (STRs). STRs occur when a short region consisting of two or more nucleotides is repeated and the occurrences are adjacent to each other. Unrelated individuals are likely to have a different number of repeats of a given STR sequence in certain regions in their DNA and thus STRs are often used for identity testing or testing between close relatives (such as paternity testing).

Garbled circuit evaluation. The basic idea behind garbled circuit evaluation is as follows (here we present only an overview of the approach and refer the reader to, e.g., [55] for technical details and security analysis): For each wire i of the Boolean circuit corresponding to f, the circuit generator creates a pair of randomly chosen labels ℓ_i^0 and ℓ_i^1 (of sufficient length that depends on the security parameter) which map to the values of 0 and 1, respectively, of this wire. Let g be a binary gate that takes two input bits and produces a single bit; also let the input wires to g have indices i and j and let the output wire have index k. Then to create a garbled representation of the gate, the circuit generator produces a truth table containing four entries of the form $\mathsf{Enc}_{\ell_i^{b_i},\ell_j^{b_j}}(\ell_k^{g(b_i,b_j)})$. Here $b_i, b_j \in \{0,1\}$ are input bits into the gate and all entries in the table are randomly permuted. Possession of two input labels $\ell_i^{b_i}$ and $\ell_j^{b_j}$ for any given values of b_i and b_j will allow for recovery of the corresponding output label $\ell_k^{g(b_i,b_j)}$ without revealing anything else. Then upon garbling all gates of the circuit, the circuit generator communicates all garbled gates, to which we collectively refer as a garbled circuit $\mathcal{G}_f$, to the circuit evaluator together with a *single* label $\ell_i^{b_i}$ for each input wire i according to the input bit b_i. The labels corresponding to the input wires of the circuit generator are simply transmitted to the evaluator, while the labels corresponding to the inputs of the circuit evaluator are communicated to the evaluator by the means of OT (see section 3.2). The knowledge of the input labels and garbled gates allows the circuit evaluator to evaluate the entire circuit in its garbled representation and obtain a label for each output wire representing the output. Then either the circuit generator sends the label pairs (in order) for all output wires to the circuit evaluator, which allows the evaluator to interpret the meaning of the labels and learn the output, or the evaluator sends computed labels to the circuit generator, which in turn allows the circuit generator to learn the result.

Naor-Pinkas OT For completeness of this work, we provide Naor-Pinkas OT protocol [61] in Figure 1.

B Additional Details

Below we summarize the overall solution with certified inputs in the presence of malicious A and B and semi-honest S as Protocol 3. For simplicity of presentation, we assume that all input bits of A and B are certified and signed in one message.

C Security Proofs

Proof of Theorem 2 We start by showing fairness and then proceed with security. The only way for A or B to learn any output is when A is satisfied with the verification of the output labels she received from B. Recall that each received label ℓ_i is checked against $H(\ell_i^b), H(\ell_i^{1-b})$ for some bit b, where H is a random oracle. The probability that this check succeeds for some ℓ_i that is not equal to ℓ_i^0 or ℓ_i^1 is negligible. Thus, A is guaranteed to possess the result of garbled circuit evaluation, at which point both parties have access to the output.

We next construct simulators for all of the (independent) adversaries $\mathcal{A}_A, \mathcal{A}_B$, and $\mathcal{A}_S$. We start with a simulator $\mathcal{S}_A$ for malicious $\mathcal{A}_A$. $\mathcal{S}_A$ runs $\mathcal{A}_A$ and simulates the remaining parties. $\mathcal{A}_A$ produces t_1 random labels ℓ_i^0 and sends them to $\mathcal{S}_A$, while $\mathcal{S}_A$ chooses Δ and sends it to $\mathcal{A}_A$. If at least one label is of an incorrect bitlength, $\mathcal{S}_A$ aborts. If $\mathcal{S}_A$ did not abort, $\mathcal{A}_A$ sends t_1 labels to $\mathcal{S}_A$. If the ith label sent by $\mathcal{A}_A$ does not correspond to one of the labels in the ith pair of labels $(\ell_i^0, \ell_i^0 \oplus \Delta)$ corresponding to $\mathcal{A}_A$'s inputs, $\mathcal{S}_A$ aborts. If $\mathcal{S}_A$ did not abort, it interprets the meaning of the input labels received from $\mathcal{A}_A$ and stores the input as x_1'. At some point $\mathcal{S}_A$ creates a random label ℓ_i for each bit of the output and sends them to $\mathcal{A}_A$. Upon $\mathcal{A}_A$'s request, $\mathcal{S}_A$ also chooses another random label ℓ_i' for each bit of the output. For each bit i of the output, $\mathcal{S}_A$ sends to $\mathcal{A}_A$ the pair $H(\ell_i), H(\ell_i')$ in a randomly permuted order. If $\mathcal{A}_A$ notifies $\mathcal{S}_A$ of successful verification of the output labels, $\mathcal{S}_A$ queries the TP for the output $f(x_1', x_2)$. For each ith bit

Input: A has private input x_1 and signature $\mathsf{Sig}(x_1)$, B has private input x_2 and $\mathsf{Sig}(x_2)$, and S has no private input.
Output: A and B learn $f(x_1, x_2)$, S learns nothing.
Protocol 3:

1. (a) S chooses $\delta \xrightarrow{R} \{0,1\}^{\kappa-1}$, $k_1 \xrightarrow{R} \{0,1\}^{\kappa}$, $k_2 \xrightarrow{R} \{0,1\}^{\kappa}$ and sets $\Delta = \delta||1$. S sends Δ and k_1 to A. S also computes labels $\ell_i^0 = \mathsf{PRF}(k_1, i)$ and $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [1, t_1]$.
   (b) A computes labels $\ell_i^0 = \mathsf{PRF}(k_1, i)$ and $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [1, t_1]$. For each bit b_i of her input, A commits $c_i = \mathsf{Com}(b_i, r_i)$ and $c'_i = \mathsf{Com}(\ell_i^{b_i}, r'_i)$ using fresh randomness r_i and r'_i. A sends to S $\mathsf{Sig}(x_1)$ and c_i, c'_i for $i \in [1, t_1]$.
   (c) A proves in ZK the statement in equation 2 using private inputs $x_1, b_1, \ldots, b_{t_1}, r_1, \ldots, r_{t_1}$. For each $i \in [1, t_1]$, A also proves in ZK the statement in equation 4 using private inputs $b_i, r_i, \ell_i^{b_i}, r'_i$.
2. S computes wire labels $\ell_i^0 = \mathsf{PRF}(k_2, i - t_1)$ and $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [t_1 + 1, m]$. S then construct garbled gates $\mathcal{G}_f$ and sends $\mathcal{G}_f$ and A's commitments c'_i for $i \in [1, t_1]$ to B.
3. S and B engage in t_2 instances of 1-out-of-2 OT as in Protocol 2 together with verification of B's input. Before B can learn labels $\ell_{t_1+i}^{b_i}$, B forms t_2 commitments $c''_i = \mathsf{Com}(b_i, r''_i)$ using fresh randomness r''_i and proves in ZK the statements in equations 2 and 3 using private input $x_2, b_1, \ldots, b_{t_2}, r''_1, \ldots, r''_{t_2}$ and b_i, r''_i, k_i, respectively. Here k_i denotes the value chosen during step 2 of the ith instance of the OT protocol.
4. A opens commitments c'_i by sending to B pairs $(\ell_i^{b_i}, r'_i)$ for i $\in [1, t_1]$. B checks whether $\mathsf{Com}(\ell_i, r'_i) = c'_i$ for each i and aborts if at least one check fails.
5. The remaining steps are the same as in Protocol 2.

b_i of the output, if $b_i = 0$, $\mathcal{S}_A$ sends to $\mathcal{A}_A$ the pair (ℓ_i, ℓ'_i), otherwise, $\mathcal{S}_A$ sends the pair (ℓ'_i, ℓ_i).

Now we examine the view of $\mathcal{A}_A$ in the real and ideal model executions and correctness of the output. After receiving the label pairs from $\mathcal{A}_A$, $\mathcal{S}_A$ performs the same checks on them as S would and thus both would abort in the same circumstances. Similarly, if $\mathcal{A}_A$ provides malformed labels for circuit evaluation, $\mathcal{S}_A$ will immediately detect this in the ideal model and abort, while B in the real world will be unable to evaluate the circuit and also abort. Otherwise, in both cases the function will be correctly evaluated on the input provided by $\mathcal{A}_A$ and B's input. In the remaining interaction, $\mathcal{A}_A$ sees only random values, which in the ideal world are constructed consistently with $\mathcal{A}_A$'s view in the real model execution. Thus, $\mathcal{A}_A$'s view is indistinguishable in the two executions.

Let us now consider malicious $\mathcal{A}_B$, for which we construct simulator $\mathcal{S}_B$ in the ideal model execution who simulates correct behavior of A and S. First, $\mathcal{S}_B$ simulates the OT. It records the input bits used by $\mathcal{A}_B$ during the simulation, which it stores as x'_2 and returns t_2 random labels to $\mathcal{A}_B$. $\mathcal{S}_B$ also sends another set of t_1 random labels to $\mathcal{S}_B$. $\mathcal{S}_B$ queries the TP for $\mathcal{A}_B$'s output $f(x_1, x'_2)$ and chooses a pair of random labels (ℓ_i^0, ℓ_i^1) for each bit i of the output. $\mathcal{S}_B$ gives to $\mathcal{A}_B$ a simulated garbled circuit (as described in [55]) so that the ith computed output label corresponds to the ith bit of $f(x_1, x'_2)$. If after circuit evaluation, $\mathcal{A}_B$ does not send the correct output labels to $\mathcal{S}_B$, $\mathcal{S}_B$ aborts the execution. Otherwise, $\mathcal{S}_B$ sends the pairs (ℓ_i^0, ℓ_i^1) to $\mathcal{A}_B$.

The only difference between the view of $\mathcal{A}_B$ in the real model and the view simulated by $\mathcal{S}_B$ in the ideal model is that $\mathcal{A}_B$ evaluates a simulated circuit in the ideal model. Computational indistinguishability of the simulated circuit follows from the security proofs of Yao's garbled circuit construction [55]. Thus, $\mathcal{A}_B$ is unable to tell the two worlds apart.

It is also straightforward to simulate the view of semi-honest $\mathcal{A}_S$ because it has no input and receives no output. □

Table 10. Performance of protocol 1 (with half-gates); work is in ms.

Function	Input size	Party	Computation			Communication		
			offl.	onl.	total	sent	recvd	total
AES	128	A	0.02	–	0.02	2KB	0.01KB	2.01KB
		B	0.42	–	0.42	182KB	0.11KB	182KB
		S	–	0.31	0.31	0.11KB	184KB	184KB
Hamming distance (bits)	2^{12}	A	0.06	–	0.06	64KB	0	64KB
		B	0.30	–	0.30	192KB	0.2KB	192KB
		S	–	0.23	0.23	0.19KB	256KB	256KB
	2^{13}	A	0.11	–	0.11	128KB	0	128KB
		B	0.65	–	0.65	381KB	0.2KB	381KB
		S	–	0.52	0.52	0.2KB	509KB	509KB
	2^{14}	A	0.22	–	0.22	256KB	0	256KB
		B	1.65	–	1.65	768KB	0.4KB	768KB
		S	–	1.39	1.39	0.4KB	1MB	1MB
Matrix multiplication ($n \times n$ ints)	4	A	0.01	–	0.01	8KB	0.06KB	8.1KB
		B	13.4	–	13.4	6.01MB	8KB	6.02MB
		S	–	9.78	9.78	8KB	6.02MB	6.02MB
	8	A	0.03	–	0.03	32KB	0.25KB	32.2KB
		B	107	–	107	48.0MB	32KB	48.1MB
		S	–	78.2	78.2	32KB	48.1MB	48.1MB
	16	A	0.11	–	0.11	128KB	1KB	129KB
		B	858	–	858	384MB	128KB	384MB
		S	–	621	621	128KB	384MB	384MB
Edit distance (chars)	32	A	0.00	–	0.00	4KB	0	4KB
		B	6.97	–	6.97	4.61MB	0.08KB	4.61MB
		S	–	5.17	5.17	0.08KB	4.6MB	4.62MB
	64	A	0.01	–	0.01	8KB	0	8KB
		B	27.9	–	27.9	18.4MB	0.9KB	18.4MB
		S	–	20.6	20.6	0.9KB	18.4KB	19.3KB
	128	A	0.01	–	0.01	16KB	0	16KB
		B	113	–	113	73.8MB	0.1KB	73.8MB
		S	–	83.6	83.6	0.11KB	73.8MB	73.8MB

D Additional Results

Table 10 provides the results of running protocol 1 using representative functions with the half-gates optimization.